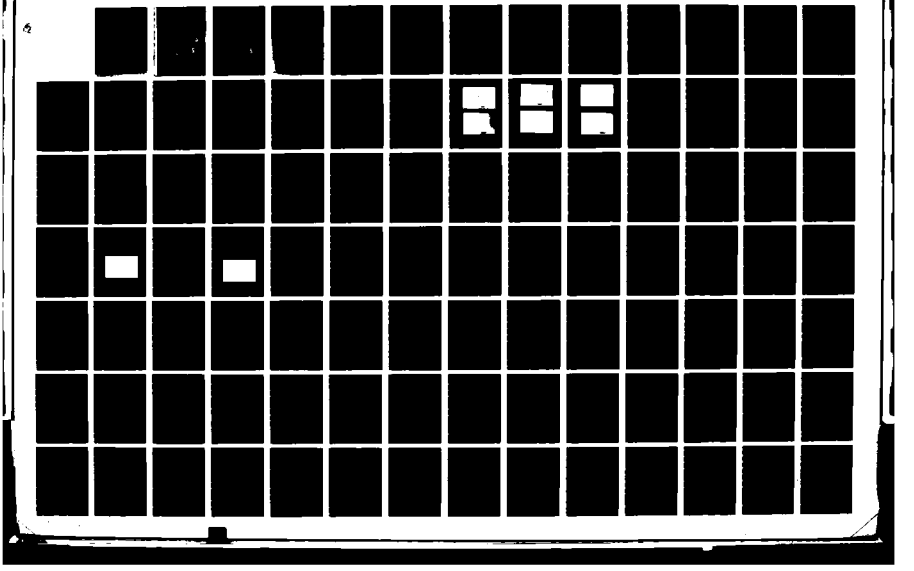
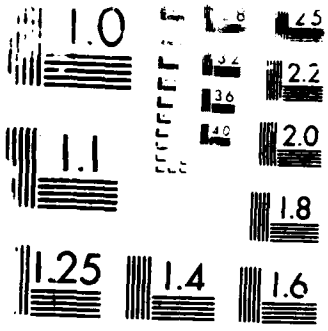
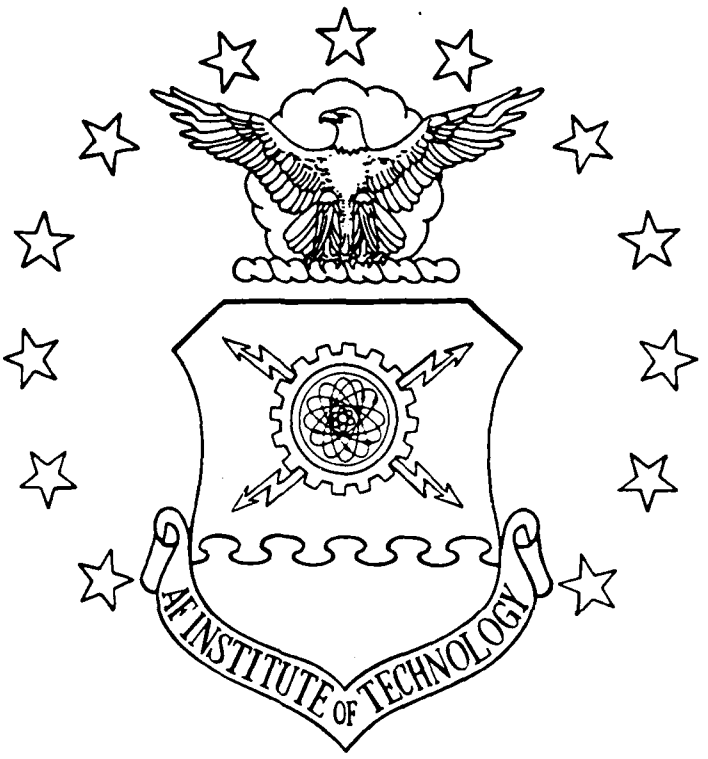
AD-A188 828 POSITION SCALE AND ROTATION INVARIANT TARGET RECOGNITION USING RANGE IMAGERY(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI

1/2

UNCLASSIFIED S E TROXEL DEC 87 AFIT/GEO/ENG/87D-3 F/G 17/9 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
3.2
3.6
4.0



POSITION, SCALE, AND ROTATION INVARIANT

TARGET RECOGNITION USING RANGE IMAGERY

THESIS

Steven E. Troxel
First Lieutenant, USAF

AFIT/GEO/ENG/87D-3

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GEO/ENG/87D-3

POSITION, SCALE, AND ROTATION INVARIANT TARGET RECOGNITION USING RANGE IMAGERY

THESIS

Steven E. Troxel
First Lieutenant, USAF

AFIT/GEO/ENG/87D-3

Approved for public release; distribution unlimited.

AFIT/GEO/ENG/87D-3

POSITION, SCALE, AND ROTATION INVARIANT TARGET RECOGNITION USING RANGE IMAGERY

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Steven E. Troxel, B.S.E.E

First Lieutenant, USAF

December 1987

Acknowledgments

There are many individuals who deserve thanks for thier support during this research project. First, I am deeply indebted to my thesis advisor, Dr. Steven K. Rogers, for his support and constant encouragement during the entire thesis effort. I also thank Dr. Rogers for giving enough rope to allow me to perform pure research and yet not enough rope to hang myself with. There is a very fine line with a time limited project. I thank Dr. Matthew Kabrisky for making research rewarding. If I told him I noticed that the sun rose in the east, I'm sure he would praise me for noticing. I would also like to thank Dr. James P. Mills for his timely suggestions and Dan Zambon for his system level support in the Information Systems Laboratory. A very special thank you goes to Capt. Dennis Ruck for all his help with ADA programming, VMS, and TROFF. This project would have been much harder without him.

Most importantly, I express deep appreciation to my wife, Shelley, and my children, Shawn and Stacey. The many hours of study time and computer time needed for this type of project requires a very special and understanding home team to see it through. I'm thankful that I was given such a team.

Finally, I dedicate this work to the memory of my mother, Dr. Marcia Hamre Troxel, for giving me the drive and teaching me about dedication to the accomplishment of a task.

Table of Contents

List of Figures

List of Tables

Abstract

This thesis explores a new approach to the recognition of tactical targets using a multifunction laser radar sensor. Targets of interest were tanks, jeeps, and trucks. Doppler images were segmented and overlaided onto a relative range image. The resultant shapes were then transformed into a position, scale, and rotation invariant (PSRI) feature space. The classifiation process used the correlation peak of the template PSRI space and the target PSRI space as features. Two classification methods were implemented: a classical distance measurement approach and a new biologically-based neural network multilayer perceptron architecture.

Both methods demonstrated classification rates near 100% with a true rotation invariance demonstrated up to 20 degrees. Neural networks were shown to have a distinct advantage in a robust environment and when a figure of merit criteria was applied.

A space domain correlation was developed using local normalization and multistage processing to locate and classify targets in high clutter and with partially occluded targets.

I. Introduction

1.1 Purpose

For the purpose of this research effort, the process of target recognition is a two-fold process of classifying and locating a target. The classification process answers the question: "Is there something in the input scene that matches what I'm looking for." The locating process answers the question of "Where in the input scene is the thing I'm looking for." The usefulness of target recognition is well documented but a truly autonomous system is still unavailable. The focus of this research effort is in the classifying area with a small section devoted to the locating process. The classification will be performed in a feature space that is invariant to changes in position, scale, and rotation. The input data is segmented targets containing gray scale intensity values which relate information about the relative range or depth changes of the target. These templates now contain 3-D information about the target which results in more classification information. The classification process will also be performed on binary templates to determine if the method of classification is making use of the added information contained in the range data.

1.2 Background

A truly useful target recognition algorithm makes few or no assumptions on the way the target will appear in a given scene.

> " The target may vary in size, shape, orientation, and illumination, or may even be partially obstructed by other objects. As a result, digital pattern recognition machines require complex algorithms for managing even a small number of the infinite possibilities of target variations, viewing angles and scene clutter that may be encountered in a typical scene"[1:2].

The current state of target recognition is not able to address all of these variations. However, much work has been done under the restriction of knowing the orientation of

the target with respect to out-of-plane rotation[1;2;3;4;5;6].

An algorithm which has shown promising results is the AFIT algorithm. The AFIT algorithm was originally proposed by Israeli Air Force Major Moshe Horev in his thesis, "Picture Correlation for Automatic Machine Recognition" [4], and later implemented on a VAX computer by Kobel and Martin [1]. The AFIT algorithm works on the assumption that we already know that the target is within the input scene. It therefore determines the scale and rotation of the target in order to correlate and locate the target. Kobel and Martin fully implemented the AFIT algorithm and tested it using visual information. Results showed that the algorithm works when the dimensions of the clutter is not similar to the target [1:77]. The algorithm works in the frequency domain and one of the reported problems that hampered recognition was shadows of the target and changes in brightness across the target. These shadows and brightness changes produced dominate spatial frequency terms which caused the scale and rotation values to differ from the theoretical. This suggests that visual information might not be the best domain in which to perform the recognition.

Visual data has many problem areas in target recognition. If the environment for recognition could be strictly controlled, visual data could prove to be adequate. However, for a recognition system to be robust in different environments (i.e. changes in sunlight) visual data varies too much to be of use. Different types of data for possible use include passive infrared or doppler and range data obtained with a laser radar. This research effort uses laser radar data for processing.

Range data is obtained either using a pulsed laser and computing time-of-flight between the transmitted and received signal or using a CW modulated beam and measuring the phase shift. A range image is obtained by using a scanning system to sweep the beam over the scene [7:206]. Therefore, assuming that the return signal is digitized into an NxN array of pixels, each pixel will contain information relating to the relative range of the sector of the input scene covered by that pixel. Now the input

scene data is unaffected by changes in ambient illumination or shadow problems and the frequency domain contains information about how the range is changing within the scene.

This concept of using the frequency information of range data for target classification appears to be an overlooked area of research. Grantham did a thesis using model based range data of tanks and did correlation of these models [8]. Tong did a thesis using true range data of tanks and trucks in background clutter to segment out possible targets from a scene [9]. The range data provided information as to where the relatively flat things with edges were located. Earlier work by Duda and others used range data to define edges and some planar surfaces with office type scenes [7;10;11]. However, no one has used the fact that the changes in range data across a target have spatial frequency information that is unique for different targets. Thus a possible area for classification. If this turns out to be a unique idea, full credit for originality must go to Dr. Steve Rogers [12].

1.3 Problem Definition

The thesis problem was to classify and locate a target within an input scene. This process must be accomplished regardless of the target's variation in position, size, and rotation, and with scenes that contain high clutter (non-random noise).

1.4 Scope of Thesis

This thesis focused on classifying and locating a target within an input scene. Laser range data was used in an attempt to show that this data contains more information than visual data and can therefore lead to better classification. The classification process was attempted both in and out of the frequency domain with each process explored to determine if it's the rate of change in the range data that produces the classification.

The Executive program written by Kobel and Martin was used for this study [1:Vol II]. This program was used unmodified to perform the many Fourier transforms and correlations needed. It was written using the Vax Ada programming language on the Vax 11/780 computer. All the new programs used for classifying and locating were also written using Vax Ada and written so as to insure compatibility with Executive.

1.5 Approach and Assumptions

The approach was to create segmented targets containing range data and transform these targets into a position, scale, and rotation invariant (PSRI) feature space to determine the proper scale and rotation relationship between template and target. Next, the proper features to be used for classification must be determined. By definition, the proper features are any features that can separate the classes of objects that are being classified. Classification methods both in and out of the frequency domain were to be attempted. Comparisons were to be made between classification with range data and classification with binary data to determine if the range data was being used and not just the shape of the object.

The following assumptions were made during the course of this study:

1) The multisensor data was supplied from a number of target scenes. Images were supplied as a digitized 256 X 256 array of numbers with each number represented as 8 binary bits. The data included visual, laser radar, and infrared pictures.

2) The out-of-plane rotation of the target was known and templates at this orientation were available. This restricts us to in - plane recognition.

3) The location, size, and in-plane rotation of the target were unknown prior to processing. This requires a position, scale and rotation invariant algorithm.

4) The potential target was presented to the classifier segmented from the input scene.

5) The recognizer as a whole had no prior knowledge about the background information of the scene.

1.6 Overview of Thesis

This thesis is structured in an order which would naturally flow if the proposed recognition algorithm were to be carried out. The proposed algorithm is as follows:

First, segment out possible targets from an input scene. This can be accomplished with either a doppler image, if the target happens to be moving [13], and/or with a range segmenter as proposed by Tong [9:Ch 3]. This segmented scene is in a binary form and is then overlayed on a laser range scene to include the range changes in the possible targets. This is so the rest of the algorithm only has to work with one region and one target at a time. The segmenting and region selection process are the topics discussed in chapter II.

Second, the target and template are transformed into a position, scale, and rotation invariant (PSRI) feature space. A correlation is performed between the two feature spaces which determines the rotation and scale relationship between the two. If necessary and available, a properly scaled and rotated template can now be chosen from a template bank. The creation of the PSRI feature space and how a correlation determines the scale and rotation relationship are the subjects of chapter III.

Third, a window around the peak of the target - template PSRI feature space correlation is presented for classification. In this domain, classification is accomplished both with standard distance measurements and with a trainable neural network. These methods of classification are the subjects of chapter IV.

Another method of classification is to take a properly scaled and rotated template back into the space domain and perform a "normalized" correlation between the target and template. The normalized correlation used was discovered during this thesis effort and will be from this point called a Goodman - Schwartz correlation. This correlation and its use in classification are the subjects of chapter V.

Finally, experimental results which compare the use of range imagery to that of binary data are presented in chapter VI with conclusions and recommendations being presented in chapter VII.

II. Segmenting Targets

The targets were presented to the classification part of the algorithm in segmented form. Segmentation was also needed to create templates to compare the targets to. When Kobel and Martin tested the AFIT algorithm using visual images, the templates were made by creating a silhouette of the target from the actual visual picture. The template was cut out by hand and made to be black on a white background. If a smaller or larger template was desired, the silhouette was moved farther or closer to the digitizing camera and a new image was formed [1]. This approach was not used to create templates which include range information since the range data was presented with a numerical range of 256 and the digitizing camera available only had a range of 16 levels. This loss of information was thought to be unsatisfactory for an algorithm that was attempting to determine the usefulness of range information. A geometric model-based approach would be a good way to collect the data in a more realistic environment where different aspect angles would need to be computed. But this approach was well beyond the scope of this thesis. Some initial testing was performed using targets that were hand segmented directly from the laser range data. Hand segmentation simply involves setting all the pixel locations that don't belong to the target equal to zero. With a 256 x 256 image, this becomes a very tedious process. By the time it became necessary to test the algorithm using many targets and templates, Dennis Ruck, a fellow AFIT student, had found a way to produce segmented targets using the doppler information of the laser radar data [13].

These segmented doppler images were in a binary form with the target pixels having a value of one and the background having a value of zero. The segmented doppler image was then multiplied by the range image to yield a template with range information. The "Ruck Doppler Segmenter" became an invaluable tool in this thesis effort. What follows now is a brief summary of Ruck's Optimum Thesholding method of segmenting doppler data along with his region detection routine which was needed for the

case of multiple targets. For more details refer to Ruck's thesis [13].

Ruck uses an optimum thresholding method described by Gonzalez and Wintz [14:325-331] which assumes that there are two principal brightness regions in an image. In a doppler image, one region corresponds to the targets (assuming a radial velocity) and the other corresponds to the background. Therefore, the histogram of the doppler image will contain two separate clusters, a target cluster and a background cluster. Ruck shows that assuming the *a priori* probabilities of the target and background are equal, and assuming that the two clusters are indeed separate, "the optimal threshold becomes :

$$T = \frac{\mu_t + \mu_b}{2} \tag{2.1}$$

where μ_t and μ_b are the means of the target and background distributions, respectively " [13].

Now its a simple matter to set each pixel with a value greater than the threshold equal to one and less than the threshold equal to zero. In the above analysis, there is no guarantee on which side of threshold the actual target pixels will fall. Ruck overcomes this obstacle by assuming that there should always be more background pixels than target pixels. Therefore, if the resulting picture contains more ones than zeros, he reversed the polarity of the segmented image between zero and one [13].

The segmented image now contains all the moving objects within the input scene in binary form. Its then necessary to determine which areas in the scene are worth further processing (ie. classification). Ruck identifies these regions by first scanning the entire segmented image and creating a list of the borders of all the clusters of pixels. If this border list was less than a threshold set at 50, the region was deemed too small to bother with. Each of the remaining lists corresponded to possible targets that needed to be classified [13]. The border lists were then separately reconstructed into binary images which could be multiplied by the complete laser range image to produce the desired laser range targets and templates.

Ruck's doppler segmenter was used for the vast majority of the experimentation in this thesis. However, it does have an obvious drawback with non-moving targets. The segmenter proposed by Tong uses a combination of laser range and infrared data to segment out the "man-made" objects out of cluttered backgrounds. Through a process of gradient operations, mask generation, and conditional neighborhood filtering, Tong was successful in producing segmented binary images of the desired targets [15]. These segmented targets tend to have a certain amount of "blobness" associated with them but the robustness of the process makes these targets worth an attempt at classification.

A detailed analysis showing the usefulness of classifying with Ruck - segmented targets versus Tong - segmented targets was not accomplished. Some Tong - segmented targets were simply thrown into the classification pot and checked to see if problems resulted. If both types of templates could be classified then each method was deemed useful.

Once a template and possible target was identified, the next step was to transform each into the PSRI feature space. This transformation was necessary whether the classification was to be performed on the correlation peak or with the actual target in the space domain. The next chapter covers the PSRI feature space.

III. The PSRI Feature Space

3.1 Creation of PSRI Space

Since the segmented target could be presented to the classifier with any position, scale or rotation characteristics, a space which is invariant to these changes is needed. Much work has been done with the F(ln r,θ) position, scale, and rotation invariant (PSRI) feature space by Casasent [3] and locally by Mayo, Horev, Kobel and Martin [1,4,16]. In this case, a PSRI feature space is not one who's features are totally invariant to position, scale and rotation but rather one who's features behave in a very predictable manner with respect to position, scale and rotation.

The position invariance is the only true invariant part of this feature space and is accomplished by taking the magnitude of the Fourier transform of the input scene. The position invariance of this transformation is indicated by the Fourier transform "shift" theorem:

$$\text{If } F\{ i(x,y) \} = I(f_x,f_y) \quad \text{then}$$

$$F\{ i(x-\alpha,y-\beta) \} = I(f_x,f_y)e^{[-j2\pi(\alpha f_x + \beta f_y)]} \tag{3.1}$$

where i(x - α,y - β) = input image shifted by α units in the x direction and by β units in the y direction

F{ } = Two dimensional Fourier transform.

Therefore, shifts in the input scene will only affect the phase portion of the Fourier transform and will have no affect on the magnitude portion. This is very nice for making the space shift invariant, but, it runs a great risk if this space is now used for classification. By throwing away the phase, we assume that information in the magnitude of the Fourier Transform is enough for classification [17].

Rotation 'invariance' is accomplished by first realizing that rotations within the input scene result in exact equal rotations in the magnitude Fourier transform plane. If the magnitude Fourier transform plane is mapped into polar coordinates, rotations in the input plane will result in linear shifts along the angle axis. The new spatial frequency

coordinates are given by:

$$f_\theta = \tan^{-1}\left[\frac{f_y}{f_x}\right] \tag{3.2}$$

and

$$f_r = (f_x^2 + f_y^2)^{1/2} \tag{3.3}$$

Kobel and Martin used only half of the magnitude Fourier transform for conversion to the polar coordinates since the magnitude spectrum has even symmetry [1:17-18].

The scale 'invariance' makes use of the following Fourier transform property :

$$F\left\{i(\frac{x}{\alpha}, \frac{y}{\beta})\right\} = |\alpha\beta|\, I(\alpha f_x, \beta f_y) \tag{3.4}$$

This property is simplified by realizing that there are not different shaped targets only the appearance of different sized targets due to how far the target is away. This type of scaling will be equal in the x and y direction and therefore $\alpha = \beta$. Since the angular axis (f_θ) was created from the ratio of the y-direction frequencies to that of the x- direction frequencies, it is not affected by uniform scaling.

The radial frequency coordinate of the scaled polar magnitude spectrum is now :

$$f_r' = [\alpha^2(f_x^2 + f_y^2)]^{1/2} \tag{3.5}$$

Taking the natural log of both the scaled and the unscaled radial frequency coordinate results in :

$$ln(f_r) = \frac{1}{2} ln(f_x^2 + f_y^2) \tag{3.6}$$

and

$$ln(f_r') = ln(\alpha) + \frac{1}{2} ln(f_x^2 + f_y^2) \tag{3.7}$$

or

$$ln(f_r') - ln(f_r) = ln(\alpha) \tag{3.8}$$

Therefore, if the radial axis is logarithmically scaled, uniform scaling of the input will result in a linear shift along the ln f_r axis. Smaller targets will shift in the positive direction and larger targets will shift in the negative direction. The properties of the PSRI

feature space are shown in figures 3.1 - 3.5.

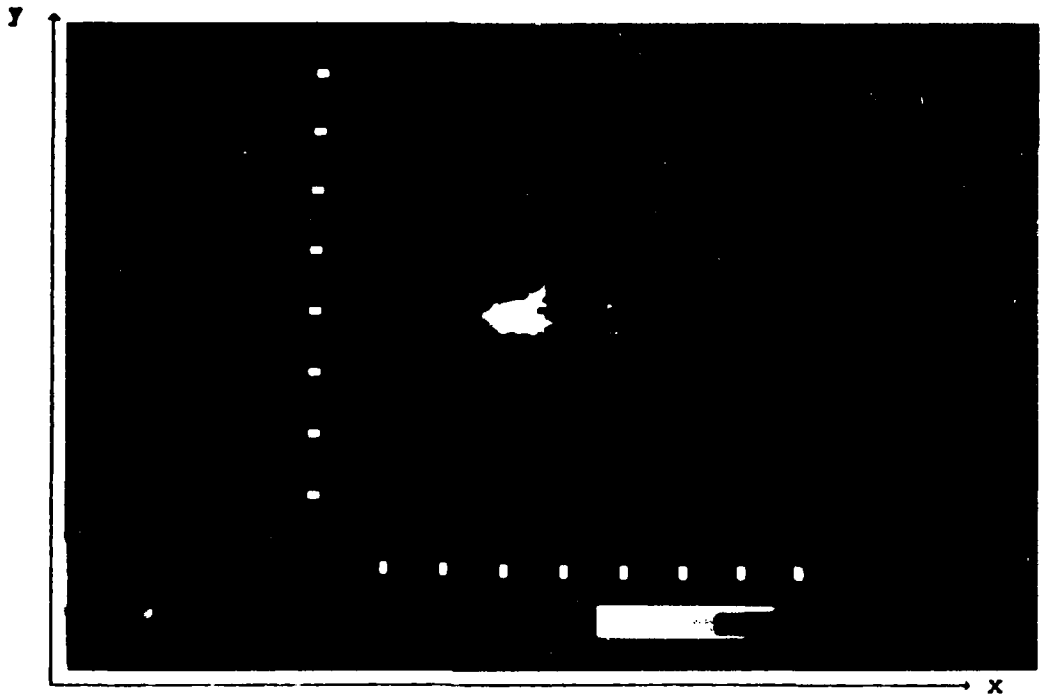


Fig 3.1
Original Template

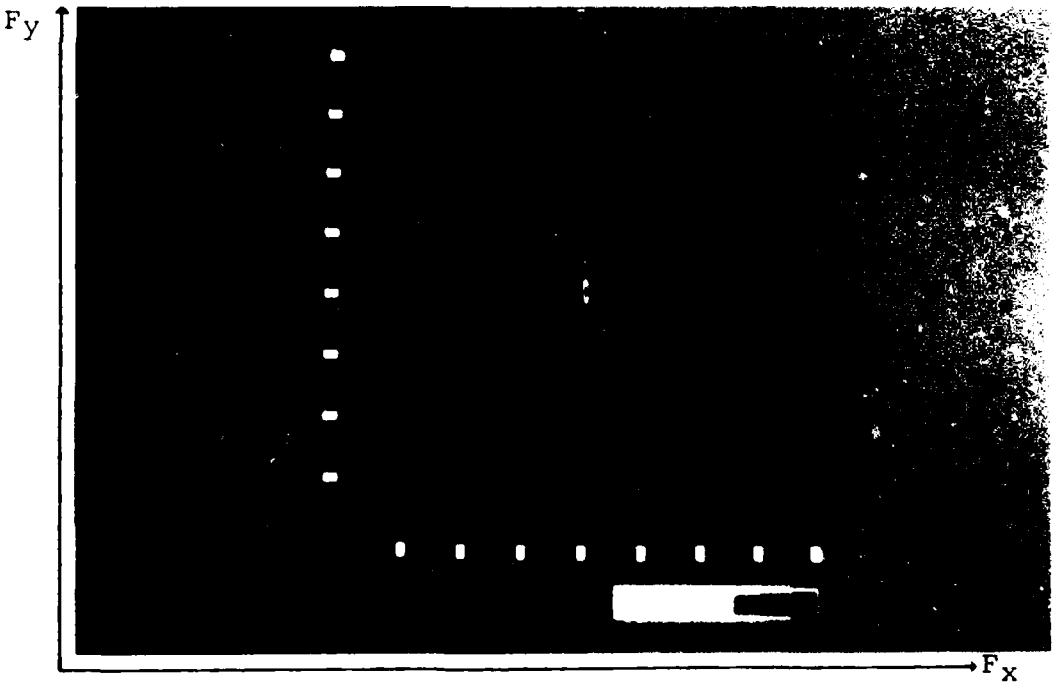


Fig 3.2
Magnitude Fourier Transform of Template
(Note: Magnitude Fourier Transform is Invariant to Shifts)

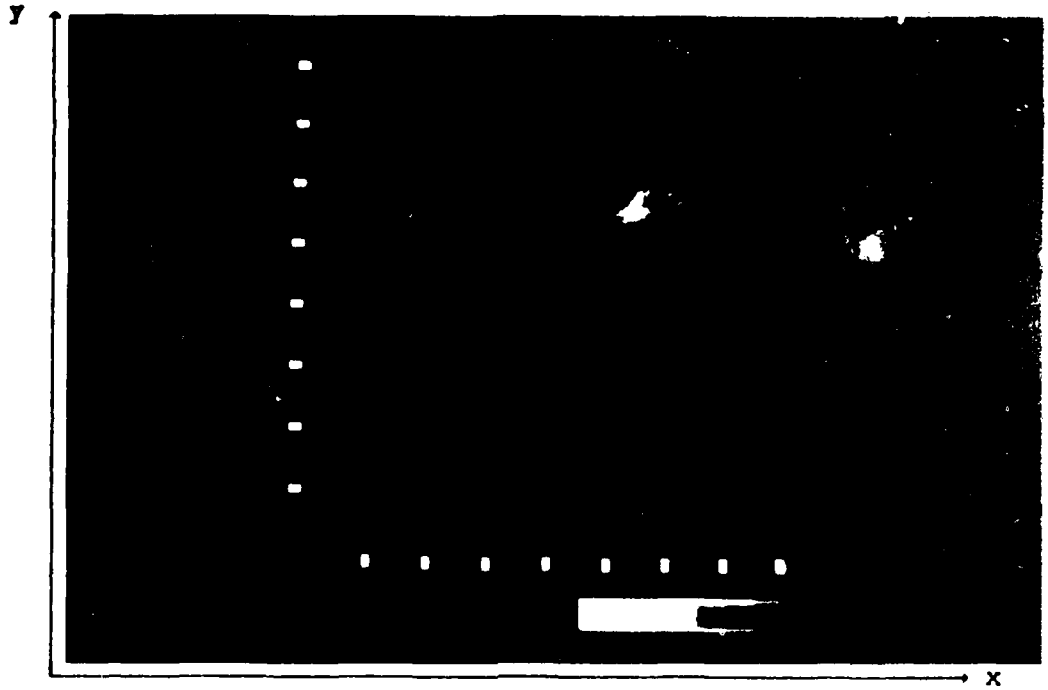


Fig 3.3
Shifted, Scaled, and Rotated Version of Template

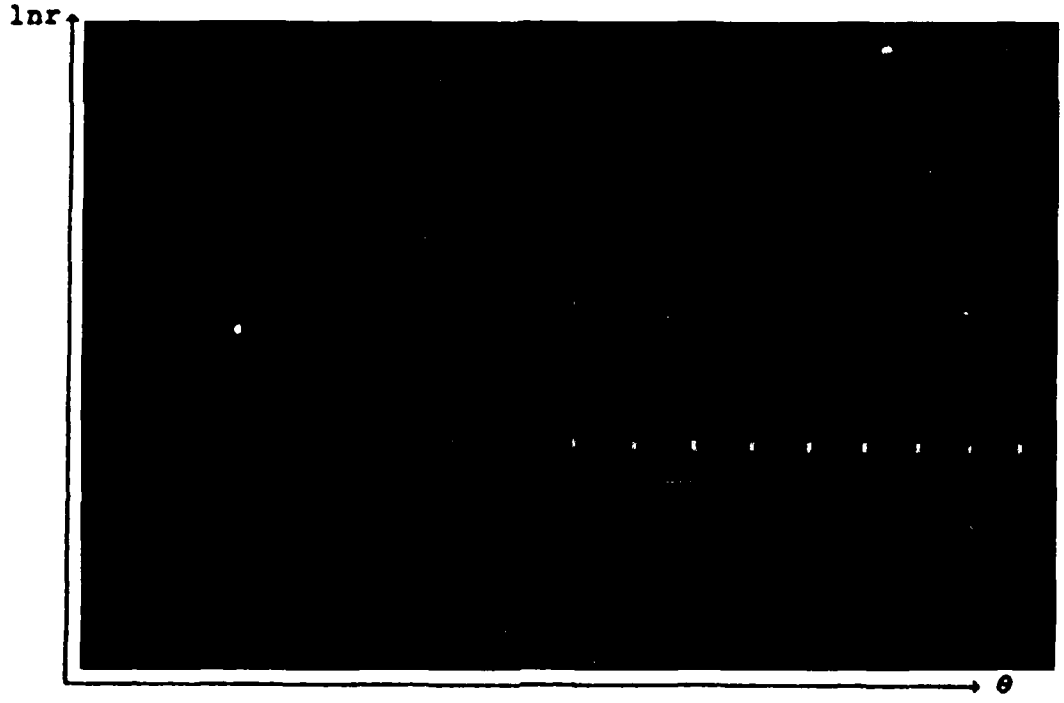


Fig 3.4
PSRI Space of Original Template

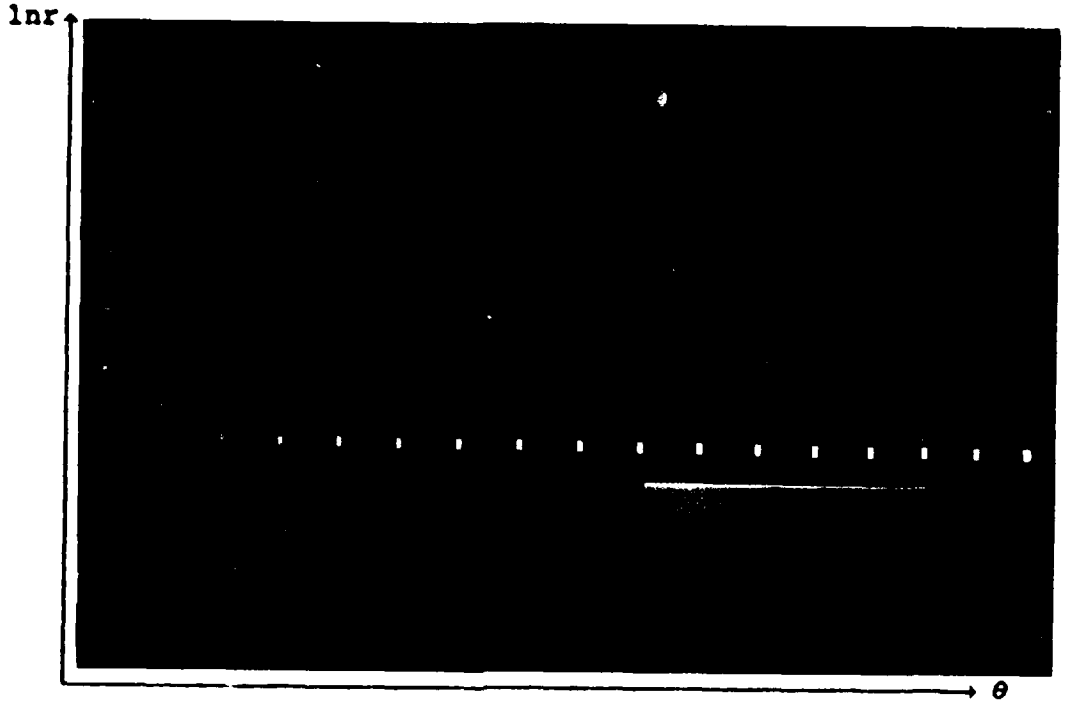


Fig 3.5
PSRI Space of Shifted, Scaled, and Rotated Template
(Note: Three operations are independent and cause a
measureable shift in the PSRI space.)

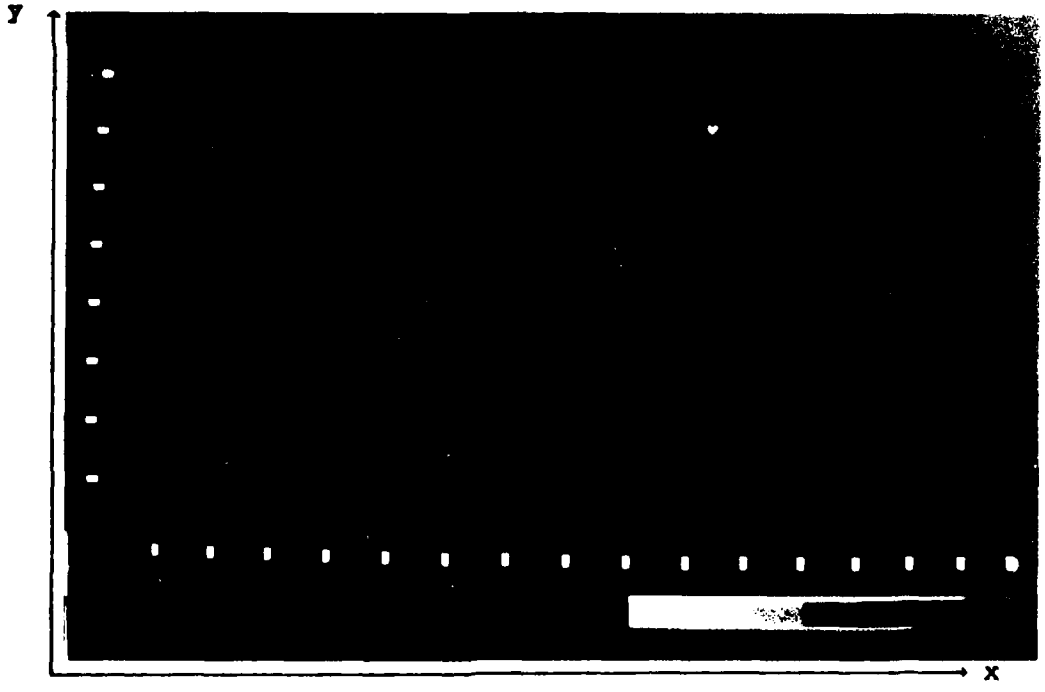


Fig 3.6
Correlation of Previous PSRI Spaces
(Note: Correlation peak indicates amount of shift
and therefore scale and rotation.)

There are a few subtle points about this feature space that need to be pointed out. First, invariance in rotation translates to linear shifts along the f_θ axis. This axis is periodic so shifts off the right end wrap around to the beginning of the left end. Second, invariance in scale translates to linear shifts along the $\ln f_r$ axis. This axis is not periodic. The $\ln f_r$ axis can be thought of as extending to infinity in both directions above and below the region chosen to be analyzed. A linear shift then means to shift this chosen region. Smaller targets causing a positive shift means that new information is pushed in from the bottom and information is lost off the top. This can cause problems in classification for large scale changes depending on how closely related the classes are to one another.

3.2 Use of PSRI Space

One of the best uses of this PSRI feature space is in determining how much a template is scaled and rotated with respect to the target. Rotations are represented by linear shifts along the f_θ axis and scale changes are represented by linear shifts along the $\ln f_r$ axis. Therefore, a correlation between the target and the template PSRI spaces will tell how much scale and rotation difference there is. The correlation plane will contain a peak value at a location representative of how much the template PSRI space had to shift to match the target PSRI space. If the target and template are the same scale and rotation, a peak at location (0,0) in the correlation plane will result. The results section of this thesis contains data which shows that shifts along the f_θ axis occur at a rate of 2.8 pixels/degree rotation and along the $\ln f_r$ axis at a rate of 30 pixels for a scale change of 2:1 (See Fig 3.6). Once the scale and rotation changes are known, a properly scaled and rotated template could be chosen from a template bank for further processing. The results section also shows how critical a properly scaled and rotated template is for classifying and locating a target. As stated in chapter I, one proposed method of classification was to analyze the peak of a template - target PSRI feature space correlation. The theory of the PSRI feature space leads to a hope that differently scaled and

rotated objects should correlate, in the PSRI feature space, to very similar peaks. Again, this is due to the fact that, in theory, the only change in the PSRI spaces has been linear shifts. If a method of grouping similar things together and then measuring the level of "togetherness" could be found, classification would result. Classification is the subject of the next chapter.

IV. Classification

4.1 Introduction

The theory of the PSRI feature space states that targets that are scaled and rotated with respect to one another should be virtually identical (except for linear shifts) in the PSRI space. Therefore, if the PSRI space of a template is correlated with a PSRI space of a scaled and rotated target, the "shape" of the correlation should be similar to the shape of an autocorrelation of the template PSRI. If a way to determine and measure these similarities could be found, then classification could be accomplished. This was the logic that was followed in the pursuit of classification.

The correlation used for all the PSRI feature spaces was the linear modified phase correlation as discussed in Kobel and Martin [1:37-39]. In this correlation, the magnitude of the templates Fourier transform is set to one during the correlation and has the effect of edge enhancing the PSRI space of the template. This effect can be thought of as a very specialized high-pass filter being applied to the magnitude transform. All Fourier transforms of real objects have large low frequency magnitudes relative to the high frequency magnitudes and the high frequency magnitudes always approach zero as the frequencies become very large. Therefore, setting all the magnitudes equal to one has the effect of attenuating the low frequencies relative to the high frequencies, thus a high-pass filter. Now, since the PSRI space of the template is edge enhanced, the correlation plane will contain a peak where the shape of the template PSRI space best matches the shape of the target PSRI space. This last statement is not generalized to all modified phase correlations but works in this case since the PSRI space is based on a magnitude Fourier transform. The energy in the PSRI space peaks and falls off nicely from these peaks without spurious high energy pockets. Without a normalization scheme, this correlation could not be expected to produce as good results with a cluttered space domain correlation. After the complete correlation was performed, the peak was found with a simple maximum value search routine and the location of the peak

very accurately defined the scale and rotation of the target with respect to the template. The peak in the correlation plane was chosen as the starting point for feature extraction.

Tallman showed that the lower three harmonics of the Fourier transform are enough to adequately discriminate handwritten letters of the alphabet [18]. These lower three harmonics, along with the DC component, result in a 7 by 7 array of numbers. Even though it's an enormous jump between defining an object in the low frequency Fourier transform space to defining an object based on some peak in the correlation plane of the PSRI spaces, this 7x7 array around the peak was the initial features for attempted recognition. Examples of these peaks are shown in figures 4.1 and 4.2.

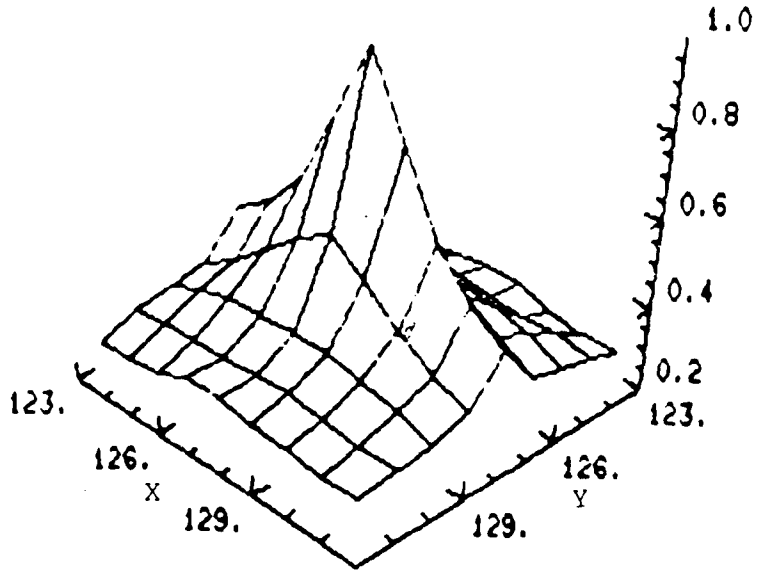


Fig 4.1 Example of an Autocorrelation Peak
(Note: This smooth fall off is present in
all the autocorrelation peaks.)

It seems appropriate at this point to momentarily digress to some general thoughts on target recognition. No ideal, all purpose features have ever been found that can always classify any target. It is reasonable to assume that such features may never be found and very well may not exist. However, many types of features have been found

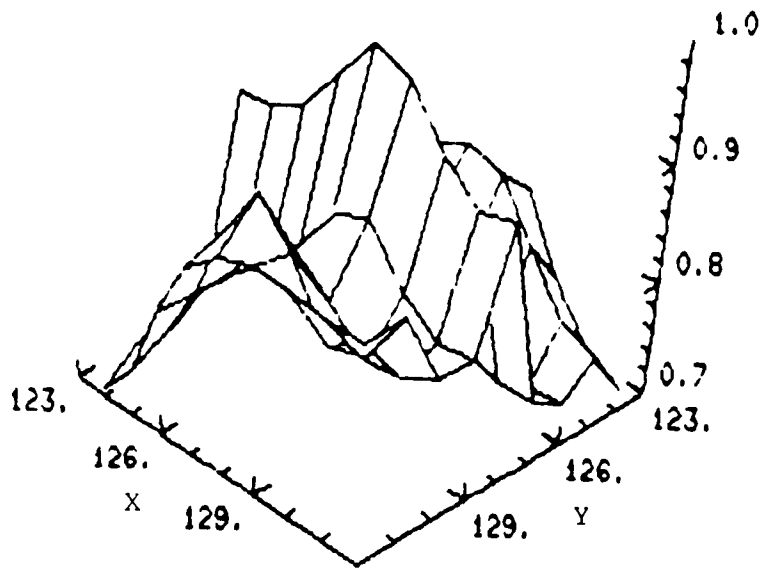


Fig 4.2 Example of a Crosscorrelation Peak
(Note: All crosscorrelation peaks lack the smooth fall off.)

and used sucessfully for recognition given a set of restrictions on such things as size, rotation, aspect angle, noise and illumination. Even human beings, which possess a remarkable recognition system, have restrictions such as distance and illumination.

There is an almost endless supply of features that can be extracted from an object. The only limitations seem to be the limitation of the imagination. However, by definition, good features are ones that result in recognition and better features are ones that result in a smaller set of restrictions. With any given set of features there is really no way of knowing if they are a good set without testing them. Even a *good* set of features might be rejected because the proper form of testing can't be found. There is much trial and error involved in the art/science of pattern recognition and any pursuit of recognition is destined to result in many more failures than successes. Therefore, the reason for choosing a 7x7 window around the correlation peak of the PSRI feature spaces for classification was based on a mixture of scientific reasoning, from the theory of the PSRI feature space, and the fact that it "felt" like it should work, since Tallman

did similar work with letters.

4.2 Correlation Peak Analysis

The testing method was to compare the window with an autocorrelation window. The autocorrelation window was a 7x7 window around the peak of the autocorrelation of a template PSRI feature space. The comparison was done by first normalizing each window by dividing each point by the square root of the sum of the squares of all the points in the window. The 7x7 array of numbers can now be thought of as a 49 dimensional vector with a length of 1. Each vector, from every correlation, now specifies a point on the surface of a 49 dimensional hypersphere. If the point representing the autocorrelation represents an exact match between the template and the target, then even though the target may be rotated and scaled differently, the correlation peak should look similar to an autocorrelation and be located very close to the true autocorrelation peak on the 49 space hypersphere. Of coarse, "very close" is an extremely relative term. The distance between the two, if measured as $D = (\sum_{i=1}^{49} (x_{1i} - x_{2i})^2)^{\frac{1}{2}}$, need only be closer than any cross correlations in order to be separable.

For example, if a target, T, has the possibility of being one of 3 classes A, B, or C, then three PSRI feature space correlations are performed, (T*A, T*B, T*C). The distances in 49 space between each of the correlation peaks and the corresponding autocorrelation peaks are calculated, (T*A - A*A, T*B - B*B, T*C - C*C). The smallest distance will then correspond to the correct match. In this thesis experiment, there are only 2 classes, autocorrelations or crosscorrelations. Without placing too many restrictions on the algorithm, it was hoped that the autocorrelation peaks would cluster together and that the crosscorrelations could be located anywhere outside this cluster. Therefore, it was only necessary to measure the distance between the correlation and a single reference autocorrelation. A threshold value could then be set to determine if the correlation was within the autocorrelation region. This assumes that the correlation

points on the 49 space hypersphere are clustered into neatly packed regions and that any wrongly matched correlation will be located farther away from the true autocorrelation than all desired properly matched correlations. However, its not guaranteed that the clustering will be so neat.

It is possible for regions to be clustered into arbitrary shapes where distances within a region are not smaller than region to region distances, strips for example. In this case, it's necessary to either find another testing method besides distances or perform some type of processing on the features which will spread the regions farther apart. Another testing method is found in the newly rediscovered field of neural networks.

Appendix A explains more about neural nets, but the general theory of the multilayer perceptron neural net is that given a set of data to train with, the net will formulate the boundaries around the regions. Once the training is accomplished, test data is supplied and the net will indicate which region it falls in. The region identification results in classification. Identification is via a numeric value of the output of the network where a value of greater than an upper threshold indicates a true and a value less than a lower threshold indicates a false. The upper threshold is usually taken to be 0.9 and the lower threshold taken to be 0.1. This separation between a true and a false value results in a very good figure of merit criteria. The figure of merit gives that warm fuzzy about how much the classification decision can be trusted [19]. Both distance measurements and neural nets were used as tests with results shown in chapter VI.

Another method of classification worked with in this thesis was performing a normalized correlation within the space domain. This method requires determining the proper scale and rotation for the template and then correlating the template with the segmented target in the space domain. The classification can now be accomplished on the magnitude of the correlation peak. As shown in chapter V, the correlation peak will be maximum when an exact point for point numerical match occurs between the target and template. Since the range information is periodic, the absolute numeric values of the

target can't be predicted. Two preprocessing methods were used in an attempt to make the target and the template have the same absolute values. Method 1 involves taking the gradient of both the target and the template before correlating. The gradient operation ignores the absolute values and assigns new values based on the numerical change between points. This has a nice "flavor" to it as it emphasizes the fact that the changes in the range data are unique for a given target. The second method is to simply add a value to each point in the template that makes the target and the template have the same average value. This is accomplished by finding the average of the points in the target and in the template and then adding the difference of the averages to each point in the template. Both methods were tried and the results are shown in chapter VI. The details of this space domain correlation are discussed in the next chapter.

V. Correlation

5.1 Introduction

Most pattern recognition algorithms use some form of correlation as the method of finally locating the target. Basic correlation involves shifting the template to all possible locations within the input scene and summing the product of all corresponding points. A basic correlation works very well for a scene in which the target energy is much greater than the background noise, where all possible targets have equal energy, and where the shape, size, and orientation of the target is very accurately known. Since very few of these type scenes exist, the basic unmodified correlation is very rarely used [19].

The main problem with using an unmodified correlation on most real scenes is that many times there is more energy in the noisy background than in the target. Therefore, the correlation peak due to the target will be much lower than the correlation peaks associated with the background. Hence, it has only limited applicability. Ideally, it's desired to have the largest peak in the correlation plane identify the location of the target. Kobel and Martin partially overcame this problem when correlating feature spaces by using a modified phase correlation [1:38].

Basically, the modified phase correlation involved correlating the template with an edge enhanced version of the input scene. This correlation was used for all the correlations of the PSRI feature spaces, and it works well when looking for a pattern in a scene that doesn't contain a great deal of rapidly changing noise. Two problems make this unsuitable for correlation with range data. First, most laser range data contains a great deal of rapidly changing noise in such things as trees and bushes. Second, since the range data is periodic from 0 to 256, the jump from 256 back to 0 will contain a large amount of energy when edge enhanced.

Another type of correlation suggested by Kobel and Martin was to first rectify the image by making the magnitude of it's Fourier transform equal to the magnitude of the

templates Fourier transform. This process enhances the frequencies associated with the template and attenuates the noise. A problem found with this type of correlation is that the frequency associated with the range gates (the spacing between the range jumps) can be very close to the dominant frequencies of the template. Therefore, when the frequencies of the template are enhanced, a large amount of energy is put into these gates. This results in correlation peaks at the areas of "correctly" spaced range gates and not at the target. These problems are not intended to state that these correlation methods will never work on range data, just to show some of the problems that need to be overcome. For someone more adept at computer programing and graphics, it may be a relatively simple task to rid the input scene of the range jumps. Also, for the newer AM/FM laser radars, where the absolute range will be available, the problem will not exist.

5.2 Goodman - Schwarz Correlation

When using a correlation for direct target classification of a segmented target as proposed in chapter IV, it becomes necessary to have an expectation of the correlation peak. A correlation method that was found to produce very desirable results is what will be referred to as a Goodman - Schwarz correlation. Although a reference for the use of this correlation was not found, the method is a simple extension of a character - recognition system discussed by Goodman [20:179-181] and is sure to have been used before. The Goodman - Schwarz correlation is accomplished entirely in the space domain using as a starting point the brute force definition of correlation. The equation for a space domain, discrete, brute force correlation is as follows:

$$R_{xy}(\alpha,\beta) = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n,m)\, y(n+\alpha, m+\beta) \tag{5.1}$$

This equation says that every point in the correlation plane is a result of the sum of a point by point multiplication of the image with a shifted template where the shift corresponds to the location of the point in the correlation plane. Again this basic correlation does very little for locating or classifying except for very simplified, highly

restrictive cases.

The main modification to this correlation is the energy normalization of the input data. Most energy normalization schemes, including the character - recognition system discussed by Goodman, are involved with image and templates of the same array dimensions and/or with the restriction of one object in the input scene, usually with relatively minor amounts of noise. With a smaller template array and an image filled with a large amount of noise and several possible targets, it becomes necessary to do a more localized normalization. The character - recognition system in Goodman takes an input image (a character) and compares it, basically using a correlation, with a series of templates. The correlation is then normalized by the division of the square root of the sum of the squares of all the points in the template (the energy of the template). Goodman shows a proof using the Schwarz inequality which states that the peak magnitude squared of this correlation is an absolute maximum when the template and the image *numerically* match. Numerically is stressed to emphasize that its the numbers that are needed to match and not just the shapes. Also, when the template and the image exactly match, the magnitude squared of the correlation peak will be equal to the sum of the squares of the template (the square of the energy). This fact gives us the expected value needed for classification.

It's a simple modification of the character - recognition system to reverse the image and template roles. Now there is one template, several inputs, and normalization is by the input energy. The shifting method in the Goodman-Schwarz correlation is the same as the brute force correlation except at each shift in the correlation, the part of the image corresponding to the location of the shifted template is considered to be the input to the modified character - recognition system. The output of the modified character - recognition system becomes the value of that shift point in the Goodman-Schwarz correlation plane. A simple peak search of the correlation plane will identify the location where the image best matched the template. If the peak value is equal to the square of

the energy, that location contains an exact match of the template. Also, its possible to determine a threshold, related to the theoretical peak value, that will result in acceptable classification.

This method would have a hard time with a small binary square template attempting to locate another small binary square in a scene that also contains large binary squares. Since the normalization only uses the energy of the image that corresponds to the location of the shifted template, all the locations in the correlation plane representing the large squares will also contain the proper maximum peak value which would indicate an exact match. However, with the use of the range data, the process was found to work very well for locating and classifying a target when the template was taken directly from the input scene (ie. same scale, rotation, aspect angle and relative range values). Also, as stated in chapter IV, the gradient operation or an average equalization can be used to allow for differences in the absolute value of the range information. In this manner, the locating and classifying was accomplished in a single step. In addition, this method has the possibility of working with partially occluded targets. The template could be broken into a number of sectors with each sector now being run through the correlation process as a separate template. The final classification would be based on the number of adjoining sectors that reached threshold, five out of nine for example. A closer examination of this process also shows that a completely parallel architecture is possible. Experimental results of all the classification methods are presented in the next chapter.

VI. Experimental Results

6.1 Introduction

This section will show and discuss the results of the various experiments conducted during this thesis effort. First will be the verification of the use of a PSRI space correlation to identify the scale and rotation of a target with respect to a given template using laser range data. This is a "get your feet wet" experiment since it uses the parts of Kobel and Martin's Executive program that will be needed for the remaining experiments. Kobel and Martin already demonstrated that this process works well with visual data [1:66-67] and it was necessary to insure that range data would also work. Next will be the classification experiment. This portion consists of three separate experiments with each portion comparing the use of range data to that of binary data for classification. The three experiments include a standard distance measurement and a neural network with data obtained from a PSRI space correlation, and a space domain Goodman - Schwarz correlation with actual space domain input data.

6.2 Identification of Scale and Rotation

This experiment was set up to verify the use of a PSRI space correlation to identify how much a target is rotated and scaled with respect to a template. The setup was to take a template and rotate it from 0 to 45 degrees in increments of 5 degrees and at each rotation, perform the PSRI space correlation with the original template. A peak search was then performed on the correlation plane with the peak location and value being recorded for analysis. The process was then repeated for a template that had first been scaled by 1/2 and then again with a 1/4 original sized template. This experiment also demonstrated which type of correlation would work best and yield the clearest correlation peaks. The Executive program contains several parameters that can be changed interactively. The modified phase or "Lin1 correlation flag" option was determined to yield the best results.

The Executive program contains a peak search routine which looks for maximum

vertical strips that are 35 pixels wide and then looks for the maximum value within this maximum strip [1:104- 105]. This process might be necessary when analyzing complete input scenes, but with using a segmented target and segmented template, a more simplified maximum value search routine gave identical results and required much less processing time. Also, because of the 35 pixel window that the Executive peak search routine uses, and the fact that small rotations cause only small shifts, a rotation of less than 6 degrees will cause the program to abort. A simple modification of the program that would include a wrap around of the data for small shifts would cure this problem.

The Executive program for rotating an image was very useful in creating the rotated templates. A good method of arbitrarily scaling a template containing range data was not found. A more detailed study is needed on how detectors collect the data and therefore, how a smaller or larger target physically changes the values of each individual detector. There should be a good mathematical approach that would come out of such a study. For this experiment, scaling by 1/2 and 1/4 was accomplished by simply sampling every other pixel for 1/2 scaling and then every other pixel again for 1/4 scaling. Tables 6.1, 6.2, and 6.3 show the results of this experiment.

This data shows that the location of the peak in the PSRI space correlation plane is a very good way to determine the scale and rotation of a target with respect to a given template. The data also indicates that rotations and scale differences can be calculated using approximate conversions of 2.8 pixels per degree rotation and 30 pixels per 2:1 scale change. Also, a true autocorrelation will yield a peak at the 0,0 location which is the lower left corner of the correlation plane. A smaller target causes the peak to shift up and rotations in the counterclockwise direction cause the peak to shift to the right.

6.3 Classification

The classification experiment was a set of several experiments. This section discusses the setup for each experiment, presents the results, and analyzes some of the results.

6.3.1 Experimental Setup

This experiment was broken into three main sections. There was classification using distance measurements, classification using a neural network, and classification using the space domain Goodman - Schwarz correlation. The classification experiments that use distance measurements and the neural network use data obtained from a 7x7 array around the peak of a PSRI space correlation. Both experiments were conducted on six sets of input data with three sets created from laser range templates and three sets created from binary templates. The three sets of files are shown in tables 6.4, 6.5, and 6.6. Each name specifies a file that contains the data of the PSRI correlation peak. The first part of the name specifies the target. The middle section indicates the rotation, if any, of the target and whether the data is range data (tmp) or binary data (btmp). The last section of the name specifies the template used. For example, R3083_r10_tmp_3195.dat indicates that this is the range data correlation peak file created using for a target, R3083 (a side view of a tank) that had been rotated by 10 degrees (r10) and for a template r3195 (a side view of a tanker truck). Set one was setup to demonstrate the classification between true autocorrelations and crosscorrelations. The true autocorrelations were of identical targets and templates. This set was used to demonstrate a "best case" scenario, and determine if there was any point in traveling farther down the chosen road. If the experiments could not classify this data, there could be no hope of classifying in a more robust situation. Set two was setup to demonstrate the classification between rotation autocorrelations and crosscorrelations. Autocorrelations were to include a number of rotated versions of the target with a fixed template. Set three was setup to test the complete robustness of the classification process. The theory was that with respect to the correlation peak, a truck is a truck is a truck and a tank is a tank is a tank. Therefore, same type objects that are at similar, but not necessarily equal, aspect angles, rotations, and scales should be able to be separated from one another. Finally, as stated above, a comparison was made between the use of range data

and the use of binary input data for classification.

The underlying goal of this thesis was to determine if range data contained an added level of information that could be useful in classification. Therefore, the exact same classification experiments were to be conducted on the laser range templates and the binary templates. However, a fundamental error was made on the first pass through this experiment. The Executive program that creates the PSRI space of the targets and templates contains a variable that allows the user to set the range of frequencies that get mapped into the PSRI space. Laser range data changes value from point to point which requires high frequency components. Also, lower frequency components are useful in determining the overall shape of an object, such as width versus height. Therefore, the decision was made to map the whole range of frequencies from 1 to 128 into the PSRI space. When the binary templates were first transformed, the range of mapped frequencies were left at the default values built into Executive, 5 to 70. With the resulting set of experimental data, no meaningful conclusions could be drawn between using laser range input and using binary input since it could not be determined whether the differences were due to the actual change in input data or the change in frequency mapping. The templates were therefore retransformed using the same 1 to 128 frequency mapping.

The final experiment in classification uses the space domain Goodman - Schwarz correlation. This experiment was performed on a much smaller set of data since the correlation by itself offers no real hope of being robust in rotation or scale. It must be assumed that the aspect angle of the target is known and can be matched. It is also assumed that the PSRI space correlation identified the rotation and scale and that a properly rotated and scaled template is available. The space domain correlation was performed on a complete unsegmented laser range scene to demonstrate the robustness with respect to clutter. Therefore, a comparison between the use of range data and binary data was not made. Also, tests were made using the Goodman - Schwarz correlation to locate partially occluded targets.

6.3.2 Distance Measurements

The distance measurement used in this experiment was a standard euclidean distance on the normalized input data. This euclidean distance was measured as $D = (\sum_{i=1}^{49}(X_{1i}-X_{2i})^2)^{\frac{1}{2}}$ where the individual points are already normalized by the square root of the sum of the squares of the input. The next question was what to use as a reference point to measure the distances from. The work with neural networks had given a good feel for the usefulness in training a classifier. For a distance measurement, the training involved reading in a number of autocorrelation examples and determining an average location for these points. This average point was then used as a reference for which all the distances in the set were measured. There was no *a priori* knowledge about the distance values and the hope was that there would be a distinct difference between the distances of the autocorrelation peaks and that of the crosscorrelation peaks. A histogram plot of the distance measurements obtained from the training data was used to determine a threshold value for classifying the test data. Tables 6.11 - 6.24 show the results of this experiment.

These results indicate that the normalized PSRI space correlation peaks of true autocorrelations (set 1) cluster very well together. Distance measurements allow for setting a threshold that can classify data with an accuracy rate near 100%, only one miss from either the range or the binary data. The results of set 2 indicates that rotations of the target with respect to the template produce little movement of the points on the 49 space hypersphere. With rotations between 0 and 40 degrees, classification of near 100% was achieved. However, when the "a tank is a tank is a tank" approach was tested (set 3), the grouping clusters are dispersed into regions that make a distance measurement virtually useless. The numbers will indicate a classification accuracy of about 65%, depending on the threshold chosen, but this is only slightly better than strictly guessing. Also, since the data is distributed very close around the threshold value,

assigning any type of figure-of-merit criteria would yield even worse results.

A comparison of the binary data results to that of the range results gives no indication that one type of data is better than the other for this type of classification. This does not imply that the range data does not contain useful information, it simply means that the testing method of distance measurements between PSRI space correlation peaks does not make use of the information that is available.

6.3.3 Neural Networks

The neural network used in this experiment was the multilayer perceptron discussed in appendix A. As discussed in appendix A, a multilayer perceptron creates decision regions from a set of training files and then classifies test data from these decision regions. The experiment using distance measurements showed that the autocorrelations in sets one and two were "reasonably" clustered together and distinctly separable from the cross correlations. These sets, therefore, provided a good testing ground to determine if the net was operating properly. For all the tests, the number of nodes for each hidden layer in the network were chosen as suggested in appendix A. All initial values for the weights and thresholds were chosen from a uniform random distribution centered at 0 with a radius of 0.5. All tests were run with an eta gain term equal to 0.25 and an alpha momentum term equal to 0.70. The results of the neural network classification are shown in Tables 6.11 - 6.24.

The results for set 1 data were obtained with a total of 1000 training file iterations. The range data was able to be trained with 50 first hidden layer nodes and 2 second hidden layer nodes while the binary data required 100 first hidden layer nodes and 2 second hidden layer nodes. Both the binary and range data results showed near perfect classification, depending on the threshold chosen. The immediate advantage seen of a neural network to that of a distance measurement for classification is in the figure-of-merit. A properly trained neural network tends to make strong decisions one way or the other.

The results for set 2 data were obtained with a total of 2000 training file iterations. Both types of data were able to be trained with 150 first hidden layer nodes and 4 second hidden layer nodes. Again, classification accuracy of approximately 95% resulted.

The results for set 3 data were obtained with a total of 45,000 training file iterations. The results changed very little after about 25,000 training file iterations but the net was allowed to run in the attempt of completely classifying the training data. This goal was not reached. With both the range data and the binary data, the network was trained with 200 first hidden layer nodes and 5 second hidden layer nodes. Note that the same two files in both the range data and the binary data case were unable to be trained. This may be an indication that with the chosen features and the method of classification, much of the dominating information is the same between range and binary data. The overall classification accuracy rate with either type of data was at 85% but when only the test data was considered, this rate drops to just below 80%. These results are very promising in that they are much better than the standard distance measurements and with a far greater figure of merit able to be assigned. The classification rate of just the same class, different image, test correlations were poor. However, with the very good classification obtained in the training data, it seems reasonable to assume that better results would be obtained with a larger data base to train with. One of the training files that was not able to be trained was a Tong segmented tanker truck that was about half the size of the Ruck segmented template. There are, of coarse, going to be limitations on the amount of variance in rotation and size. More study needs to be done to determine exactly what these limitations are. Also, it may be important to keep as much similarity as possible between the template and target so Tong data may work better when correlated with other Tong data.

6.3.4 Space Domain Correlation

The final experiment conducted in this thesis was an attempt at locating and classifying a target using the Goodman-Schwarz space domain correlation as discussed in

chapter V. This was a relatively simple experiment that was set up to demonstrate the theory and determine the usefulness in a cluttered environment. First, a range image was created that contained multiple targets. The targets within the image had to be ones for which good templates were available. Such an image was not available so templates were overlaid onto a range image. The final range image contained 3 targets, 3033 (a side tank), 3028 (a front tank), and 3190_p165 (a side tanker). The range image is shown in fig 6.1. A value of 25 was then added to each pixel in each of the templates so as to insure different absolute values. Some care had to be taken to insure that neither the targets or templates contained range jumps. Range jumps cause large values when the gradient operation is performed and therefore cause havoc in the correlation.

Fig 6.1
Range Image for Goodman - Schwarz Correlation Test

The first addition to the previously described Goodman - Schwarz correlation was to break the template into 9 sections and correlate with each section, one at a time. The original reason for doing this was to save on computation time. A subsequent section would only be applied if the result of the previous sections was above a threshold set for

those sections. The idea for using this multiple stage template matching system was taken from Vanderburg and Rosenfeld [21]. The order in which the template subsections were chosen is shown in figure 6.2.

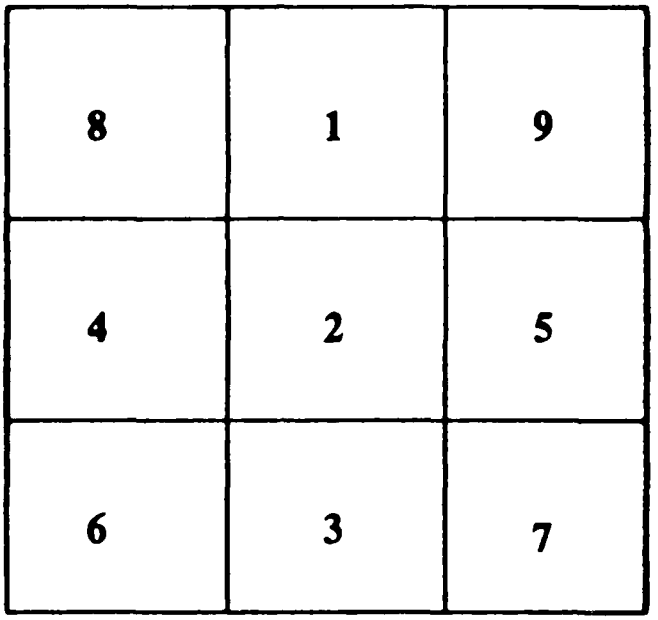


Fig 6.2
Order of Template Subsections

The second addition was to first smooth the data of both the image and the template. The thought was that in a real situation, there would be an unpredictable amount of noise in the image that would cause the target to differ from the template. It was thought that in a real scene, no single pixel should be outside the range of adjoining pixels except for some sharp edges. If these edges get slightly rounded in the process of attenuating a good deal of spurious noise, its still worth the effort. In this experiment, the template contained the same noise as the target so the true usefulness of this process could not be tested.

The third addition was that after the gradient operation was complete, a two pixel border around the template was eliminated. The reason for this was that since the background of the scene was unknown, there was no hope of matching the borders of a

gradient template. So, the correlation was performed with the interior of the gradient template.

Each template properly located the proper target using the gradient operation method. However, the peak values only achieved about 20% of the possible maximum threshold. This corresponds to entering 0.8 for the sum threshold response in the Goodman - Schwarz program. Without further study, it would be hard to assign a threshold for the purpose of classification. Using the average equalization method, the two large targets were located using a threshold criteria of 98% (0.02 in the Goodman - Schwarz program). However, the smaller target could not be located. This could be a limitation on the projected area of a target (ie. number of pixels with which to classify with). The multistage correlation allowed for much faster processing time, reducing the total number of calculations by over 80%.

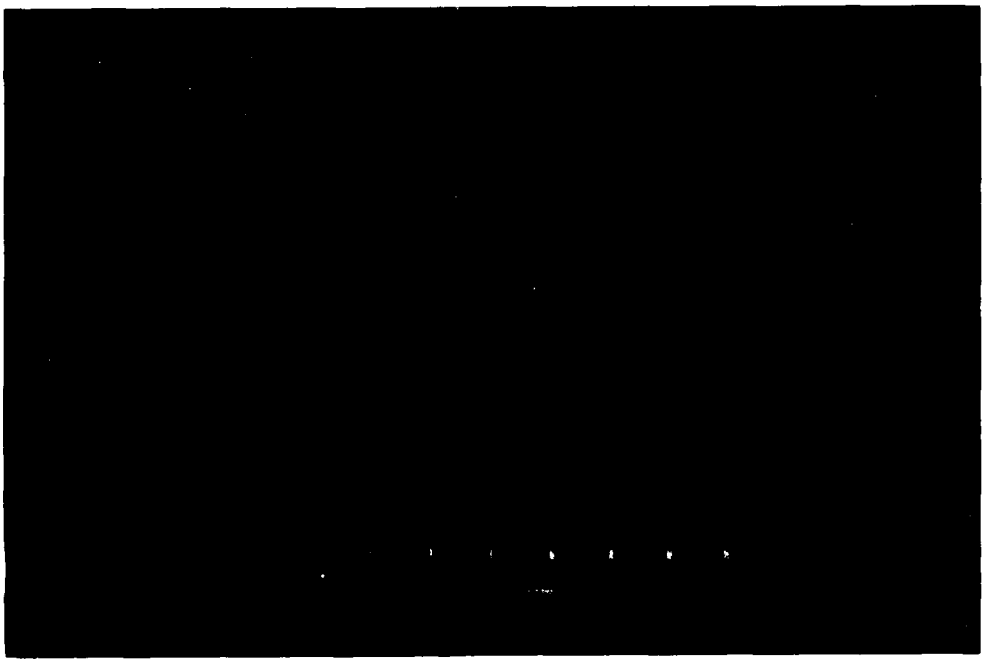


Fig 6.3
Range Image for Testing with Partially Occluded Targets

Finally, a test was performed to determine if this multistage Goodman - Schwarz correlation could be used to locate partially occluded targets. The partially occluded

targets were created by strategically placing blocks into the previous range picture. The resulting picture is shown in fig 6.3. The theory was to set the sum threshold equal to one so that all sectors would be used at every shift location. A separate threshold was then set for the sectors. This threshold would determine if a particular sector was to be counted as a match. A counter was maintained to track how many sectors at each location reached above the threshold. After the complete correlation was run, the shift locations with the top number of sectors would specify the location of the target.

Since previous problems were encountered using the gradient method, only the average equalization method was used in this test. The two large targets were located using this method but the small target was unable to be found. For the target in the lower left corner, only three shift locations in the scene had six sectors reach above a threshold of 98% (0.02 sub tolerance) and all three were located around the center of the target. For the target in the middle of the image, two shift locations had five sectors reach above 99%, both around the center of the target. The partially occluded tanker truck in the upper right corner could not be located at any threshold. This was expected since the total target couldn't be found. This target is relatively small and with the chosen block, has about 35% of its sector type information destroyed. At a threshold of 99% there were over 1000 shift locations that had six sector matches, none of those were on the target. Since the target is relatively flat, the small sectors allowed many locations within the image to be matched to the target.

This research effort contains some very interesting and promising results, however, there are still many areas that need to be further explored. Conclusions reached and recommendations for further research are discussed in the next chapter.

Table 6.1
Correlation Peak Location and Values for
Template and Same Scale Rotated Targets
(Note: Rotation shifts 2.8 pixels / degree rotation.)

File Name	Peak Location	Peak Value
R3083_tmp_3083.dat	0,0	145780
R3083_r5_tmp_3083.dat	15,0	113786
R3083_r10_tmp_3083.dat	28,0	114947
R3083_r15_tmp_3083.dat	42,0	114827
R3083_r20_tmp_3083.dat	56,0	113458
R3083_r25_tmp_3083.dat	70,0	114292
R3083_r30_tmp_3083.dat	84,0	115316
R3083_r35_tmp_3083.dat	99,0	114628
R3083_r40_tmp_3083.dat	112,0	114944
R3083_r45_tmp_3083.dat	124,0	113736

Table 6.2
Correlation Peak Location and Values for
Template and 1/2 Scaled Rotated Targets
(Note: Scale shifts 32 pixels for a 2:1 scale change.)

File Name	Peak Location	Peak Value
R3083_s2_tmp_3083.dat	0,32	57692
R3083_s2r5_tmp_3083.dat	15,32	54486
R3083_s2r10_tmp_3083.dat	29,32	53069
R3083_s2r15_tmp_3083.dat	43,32	54784
R3083_s2r20_tmp_3083.dat	57,32	53173
R3083_s2r25_tmp_3083.dat	70,32	53290
R3083_s2r30_tmp_3083.dat	85,32	54218
R3083_s2r35_tmp_3083.dat	98,32	54785
R3083_s2r40_tmp_3083.dat	113,32	54686
R3083_s2r45_tmp_3083.dat	126,32	55389

Table 6.3
Correlation Peak Location and Values for
Template and 1/4 Scaled Rotated Targets
(Note: Both scale and rotation can be calculated.)

File Name	Peak Location	Peak Value
R3083_s4_tmp_3083.dat	0,62	53439
R3083_s4r5_tmp_3083.dat	15,62	43481
R3083_s4r10_tmp_3083.dat	27,61	45756
R3083_s4r15_tmp_3083.dat	42,62	47401
R3083_s4r20_tmp_3083.dat	57,62	49740
R3083_s4r25_tmp_3083.dat	69,62	51297
R3083_s4r30_tmp_3083.dat	83,62	46519
R3083_s4r35_tmp_3083.dat	101,62	48028
R3083_s4r40_tmp_3083.dat	110,61	47040
R3083_s4r45_tmp_3083.dat	125,62	51649

Table 6.4
Files for Set 1 Experiments

File Name	Image	Template
Training Class 1		
R3028_tmp_3028.dat	Front Tank	Front Tank
R3074_tmp_3074.dat	Side Tank	Side Tank
R3066_tmp_3066.dat	Front Tank	Front Tank
R3088_tmp_3088.dat	Front Tank	Front Tank
R3083_tmp_3083.dat	Side Tank	Side Tank
R3195_tmp_3195.dat	Side Tanker	Side Tanker
R3197_tmp_3197.dat	Front Tank	Front Tank
R3215_tmp_3215.dat	Front Jeep	Front Jeep
R3197tr_tmp_3197tr.dat	Side Tanker	Side Tanker
Training Class 2		
R3028_r10_tmp_3195.dat	Front Tank	Side Tanker
r3083_r10_tmp_3195.dat	Side Tank	Side Tanker
R3195_r10_tmp_3028.dat	Side Tanker	Front Tank
R3033_tmp_3028.dat	Side Tank	Front Tank
R3190_p47_tmp_3090.dat	Side Tanker	Front Tank
R3033_tmp_3195.dat	Side Tank	Side Tanker
R3190_p165_tmp_3090.dat	Side Tanker	Front Tank
R3090_r10_tmp_3195.dat	Front Tank	Side Tanker
R3197tr_tmp_3066.dat	Side Tanker	Front tank
Test Class 1		
R3042_tmp_3042.dat	Side Tank	Side Tank
R3051_tmp_3051.dat	Front Tank	Front Tank
R3090_tmp_3090.dat	Front Tank	Front Tank
R3190_p165_tmp_3190_p165.dat	Side Tanker	Side Tanker
R3190_p221_tmp_3190_p221.dat	Side Tanker	Side Tanker
R3190_p47_tmp_3190_p47.dat	Side Tanker	Side Tanker
R3195Tank_tmp_3195Tank.dat	Side Tank	Side Tank
R3190_T227_tmp_3190_t227.dat	Side Tank	Side Tank
R3033_tmp_3033.dat	Side Tank	Side Tank
Test Class 2		
R3051_tmp_3195.dat	Front Tank	Side Tanker
R3195_r10_tmp_3028.dat	Side Tanker	Front Tank
R3195Tank_tmp_3028.dat	Side Tank	Front Tank
R3197tr_tmp_3028.dat	Side Tanker	Front Tank
R3197tr_tmp_3083.dat	Side Tanker	Side Tank
R3190_p221_tmp_3083.dat	Side Tanker	Side Tank
R3090_tmp_3083.dat	Front Tank	Side Tank
R3074_tmp_3195.dat	Side Tank	Side Tanker
R3042_tmp_3195.dat	Side Tank	Side Tanker

Table 6.5
Training Files for Set 2 Experiments

File Name	Image	Template
Training Class 1		
R3028_tmp_3028.dat	Front Tank	Front Tank
R3028_r40_tmp_3028.dat	Rotated Front Tank	Front Tank
R3066_tmp_3066.dat	Front Tank	Front Tank
R3066_r40_tmp_3066.dat	Rotated Front Tank	Front Tank
R3074_tmp_3074.dat	Side Tank	Side Tank
R3074_r40_tmp_3074.dat	Rotated Side Tank	Side Tank
R3088_tmp_3088.dat	Front Tank	Front Tank
R3088_r40_tmp_3088.dat	Rotated Front Tank	Front Tank
R3083_tmp_3083.dat	Side Tank	Side Tank
R3083_r40_tmp_3083.dat	Rotated Side Tank	Side Tank
R3195_tmp_3195.dat	Side Tanker	Side Tanker
R3195_r40_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3042_tmp_3042.dat	Side Tank	Side Tank
R3042_r40_tmp_3042.dat	Rotated Side Tank	Side Tank
Training Class 2		
R3028_R10_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_tmp_3195.dat	Front Tank	Side Tanker
R3066_tmp_3083.dat	Front Tank	Side Tank
R3074_tmp_3195.dat	Side Tank	Side Tanker
R3074_tmp_3090.dat	Side Tank	Front Tank
R3088_tmp_3190_p47.dat	Front Tank	Side Tanker
R3088_tmp_3083.dat	Front Tank	Side Tank
R3083_r10_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r10_tmp_3028.dat	Rotated Side Tank	Front Tank
R3195_r10_tmp_3028.dat	Rotated Side Tanker	Front Tank
R3042_tmp_3195.dat	Side Tank	Side Tanker
R3042_tmp_3090.dat	Side Tank	Front Tank

Table 6.6
Test Files for Set 2 Experiments

File Name	Image	Template
Test Class 1		
R3028_r10_tmp_3028.dat	Rotated Front Tank	Front Tank
R3028_r20_tmp_3028.dat	Rotated Front Tank	Front Tank
R3028_r30_tmp_3028.dat	Rotated Front Tank	Front Tank
R3066_r10_tmp_3066.dat	Rotated Front Tank	Front Tank
R3066_r20_tmp_3066.dat	Rotated Front Tank	Front Tank
R3066_r30_tmp_3066.dat	Rotated Front Tank	Front Tank
R3074_r10_tmp_3074.dat	Rotated Side Tank	Side Tank
R3074_r20_tmp_3074.dat	Rotated Side Tank	Side Tank
R3074_r30_tmp_3074.dat	Rotated Side Tank	Side Tank
R3088_r10_tmp_3088.dat	Rotated Front Tank	Front Tank
R3088_r20_tmp_3088.dat	Rotated Front Tank	Front Tank
R3088_r30_tmp_3088.dat	Rotated Front Tank	Front Tank
R3083_r10_tmp_3083.dat	Rotated Side Tank	Side Tank
R3083_r20_tmp_3083.dat	Rotated Side Tank	Side Tank
R3083_r30_tmp_3083.dat	Rotated Side Tank	Side Tank
R3195_r10_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3195_r20_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3195_r30_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3042_r10_tmp_3042.dat	Rotated Front Tank	Front Tank
R3042_r20_tmp_3042.dat	Rotated Front Tank	Front Tank
R3042_r30_tmp_3042.dat	Rotated Front Tank	Front Tank
Test Class 2		
R3028_r40_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_r40_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_r20_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_r40_tmp_3083.dat	Rotated Front Tank	Side Tank
R3066_r20_tmp_3083.dat	Rotated Front Tank	Side Tank
R3074_r40_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3074_r20_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3074_r40_tmp_3090.dat	Rotated Side Tank	Front Tank
R3074_r20_tmp_3090.dat	Rotated Side Tank	Front Tank
R3088_r20_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3088_r40_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3088_r20_tmp_3083.dat	Rotated Front Tank	Side Tank
R3088_r40_tmp_3083.dat	Rotated Front Tank	Side Tank
R3042_r20_tmp_3090.dat	Rotated Side Tank	Front Tank
R3042_r40_tmp_3090.dat	Rotated Side Tank	Front Tank
R3042_r20_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3042_r40_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r20_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r40_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r20_tmp_3028.dat	Rotated Side Tank	Front Tank
R3083_r40_tmp_3028.dat	Rotated Side Tank	Front Tank
R3195_r40_tmp_3028.dat	Rotated Side Tanker	Front Tank

Table 6.7
Class 1 Training Files for Set 3 Experiments

File Name	Image	Template
Training Class 1		
R3197tr_tmp_3195.dat	Side Tanker	Side Tanker
R3190_p165_tmp_3195.dat	Side Tanker	Side Tanker
R3190_p221_tmp_3195.dat	Side Tanker	Side Tanker
R3074_tmp_3083.dat	Side Tank	Side Tank
R3051_tmp_3090.dat	Front Tank	Front Tank
R3042_tmp_3083.dat	Side Tank	Side Tank
R3090_r40_tmp_3028.dat	Rotated Front Tank	Front Tank
R3090_tmp_3028.dat	Front Tank	Front Tank
R3033_r40_tmp_3083.dat	Rotated Side Tank	Side Tank
R3033_r10_tmp_3083.dat	Rotated Side Tank	Side Tank
R3033_tmp_3083.dat	Side Tank	Side Tank
R3028_tmp_3028.dat	Front Tank	Front Tank
R3028_r40_tmp_3028.dat	Rotated Front Tank	Front Tank
R3066_tmp_3066.dat	Front Tank	Front Tank
R3066_r40_tmp_3066.dat	Rotated Front Tank	Front Tank
R3074_tmp_3074.dat	Side Tank	Side Tank
R3074_r40_tmp_3074.dat	Rotated Side Tank	Side Tank
R3088_tmp_3088.dat	Front Tank	Front Tank
R3088_r40_tmp_3088.dat	Rotated Front Tank	Front Tank
R3083_tmp_3083.dat	Side Tank	Side Tank
R3083_r40_tmp_3083.dat	Rotated Side Tank	Side Tank
R3195_tmp_3195.dat	Side Tanker	Side Tanker
R3195_r40_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3042_tmp_3042.dat	Side Tank	Side Tank
R3042_r40_tmp_3042.dat	Rotated Side Tank	Side Tank

Table 6.8
Class 2 Training Files for Set 3 Experiments

File Name	Image	Template
Training Class 2		
R3197TR_tmp_3028.dat	Side Tanker	Front Tank
R3197TR_tmp_3083.dat	Side Tanker	Front Tank
R3190_p221_tmp_3090.dat	Side Tanker	Front Tank
R3190_p221_tmp_3083.dat	Side Tanker	Side Tank
R3190_p165_tmp_3090.dat	Side Tanker	Front Tank
R3190_p165_tmp_3083.dat	Side Tanker	Side Tank
R3051_tmp_3195.dat	Front Tank	Side Tanker
R3051_tmp_3083.dat	Front Tank	Side Tank
R3090_tmp_3083.dat	Front Tank	Side Tank
R3090_tmp_3195.dat	Front Tank	Side Tanker
R3033_tmp_3028.dat	Side Tank	Front Tank
R3033_tmp_3195.dat	Side Tank	Side Tanker
R3028_R10_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_tmp_3195.dat	Front Tank	Side Tanker
R3066_tmp_3083.dat	Front Tank	Side Tank
R3074_tmp_3195.dat	Side Tank	Side Tanker
R3074_tmp_3090.dat	Side Tank	Front Tank
R3088_tmp_3195.dat	Front Tank	Side Tanker
R3088_tmp_3083.dat	Front Tank	Side Tank
R3083_r10_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r10_tmp_3028.dat	Rotated Side Tank	Front Tank
R3195_r10_tmp_3028.dat	Rotated Side Tanker	Front Tank
R3042_tmp_3195.dat	Side Tank	Side Tanker
R3042_tmp_3090.dat	Side Tank	Front Tank

Table 6.9
Class 1 Test Files for Set 3 Experiments

File Name	Image	Template
Test Class 1		
R3190_tmp_3083.dat	Side Tank	Side Tank
R3088_tmp_3090.dat	Front Tank	Front Tank
R3035_tmp_3083.dat	Side Tank	Side Tank
R3066_tmp_3090.dat	Front Tank	Front Tank
R3190_p47_tmp_3195.dat	Side Tanker	Side Tanker
R3197TR_r40_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3195TANK_tmp_3083.dat	Side Tank	Side Tank
R3195TANK_r40_tmp_3083.dat	Rotated Side Tank	Side Tank
R3090_r30_tmp_3028.dat	Rotated Front Tank	Front Tank
R3090_r20_tmp_3028.dat	Rotated Front Tank	Front Tank
R3090_r10_tmp_3028.dat	Rotated Front Tank	Front Tank
R3033_r30_tmp_3083.dat	Rotated Side Tank	Side Tank
R3033_r20_tmp_3083.dat	Rotated Side Tank	Side Tank
R3028_r10_tmp_3028.dat	Rotated Front Tank	Front Tank
R3028_r20_tmp_3028.dat	Rotated Front Tank	Front Tank
R3028_r30_tmp_3028.dat	Rotated Front Tank	Front Tank
R3066_r10_tmp_3066.dat	Rotated Front Tank	Front Tank
R3066_r20_tmp_3066.dat	Rotated Front Tank	Front Tank
R3066_r30_tmp_3066.dat	Rotated Front Tank	Front Tank
R3074_r10_tmp_3074.dat	Rotated Side Tank	Side Tank
R3074_r20_tmp_3074.dat	Rotated Side Tank	Side Tank
R3074_r30_tmp_3074.dat	Rotated Side Tank	Side Tank
R3088_r10_tmp_3088.dat	Rotated Front Tank	Front Tank
R3088_r20_tmp_3088.dat	Rotated Front Tank	Front Tank
R3088_r30_tmp_3088.dat	Rotated Front Tank	Front Tank
R3083_r10_tmp_3083.dat	Rotated Side Tank	Side Tank
R3083_r20_tmp_3083.dat	Rotated Side Tank	Side Tank
R3083_r30_tmp_3083.dat	Rotated Side Tank	Side Tank
R3195_r10_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3195_r20_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3195_r30_tmp_3195.dat	Rotated Side Tanker	Side Tanker
R3042_r10_tmp_3042.dat	Rotated Front Tank	Front Tank
R3042_r20_tmp_3042.dat	Rotated Front Tank	Front Tank
R3042_r30_tmp_3042.dat	Rotated Front Tank	Front Tank

Table 6.10
Class 2 Test Files for Set 3 Experiments

File Name	Image	Template
Test Class 2		
R3195TANK_tmp_3028.dat	Side Tank	Front Tank
R3195TANK_tmp_3195.dat	Side Tank	Side Tanker
R3197TR_tmp_3090.dat	Side Tanker	Front Tank
R3197TR_tmp_3083.dat	Side Tanker	Side Tank
R3190_p47_tmp_3090.dat	Side Tanker	Front Tank
R3190_p47_tmp_3083.dat	Side Tanker	Side Tank
R3190_T227_tmp_3090.dat	Side Tank	Front Tank
R3190_T227_tmp_3195.dat	Side Tank	Side Tanker
R3035_tmp_3090.dat	Side Tank	Front Tank
R3035_tmp_3195.dat	Side Tank	Side Tanker
R3090_r10_tmp_3083.dat	Rotated Front Tank	Side Tank
R3090_r10_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3090_r20_tmp_3083.dat	Rotated Front Tank	Side Tank
R3090_r30_tmp_3083.dat	Rotated Front Tank	Side Tank
R3090_r20_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3090_r30_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3033_r20_tmp_3028.dat	Rotated Side Tank	Front Tank
R3033_r30_tmp_3028.dat	Rotated Side Tank	Front Tank
R3033_r20_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3033_r30_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3028_r40_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_r40_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_r20_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3066_r40_tmp_3083.dat	Rotated Front Tank	Side Tank
R3066_r20_tmp_3083.dat	Rotated Front Tank	Side Tank
R3074_r40_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3074_r20_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3074_r40_tmp_3090.dat	Rotated Side Tank	Front Tank
R3074_r20_tmp_3090.dat	Rotated Side Tank	Front Tank
R3088_r20_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3088_r40_tmp_3195.dat	Rotated Front Tank	Side Tanker
R3088_r20_tmp_3083.dat	Rotated Front Tank	Side Tank
R3088_r40_tmp_3083.dat	Rotated Front Tank	Side Tank
R3042_r20_tmp_3090.dat	Rotated Side Tank	Front Tank
R3042_r40_tmp_3090.dat	Rotated Side Tank	Front Tank
R3042_r20_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3042_r40_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r20_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r40_tmp_3195.dat	Rotated Side Tank	Side Tanker
R3083_r20_tmp_3028.dat	Rotated Side Tank	Front Tank
R3083_r40_tmp_3028.dat	Rotated Side Tank	Front Tank
R3195_r40_tmp_3028.dat	Rotated Side Tanker	Front Tank

Table 6.11
Classification Results for Set 1 Range Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 1			
R3028_tmp_3028.dat	0.0724	0.9631	0.0364
R3074_tmp_3074.dat	0.0388	0.9629	0.0366
R3066_tmp_3066.dat	0.1199	0.9631	0.0364
R3088_tmp_3088.dat	0.0783	0.9631	0.0364
R3083_tmp_3083.dat	0.0806	0.9625	0.0370
R3195_tmp_3195.dat	0.1298	0.9546	0.0448
R3197_tmp_3197.dat	0.0666	0.9631	0.0364
R3215_tmp_3215.dat	0.1088	0.9631	0.0363
R3197tr_tmp_3197tr.dat	0.1365	0.9143	0.0848
Training Class 2			
R3028_r10_tmp_3195.dat	0.2721	0.0399	0.9604
R3083_r10_tmp_3195.dat	0.2756	0.0408	0.9595
R3195_r10_tmp_3028.dat	0.2355	0.0739	0.9264
R3033_tmp_3028.dat	0.2129	0.0805	0.9198
R3190_p47_tmp_3090.dat	0.2144	0.0521	0.9482
R3033_tmp_3195.dat	0.2876	0.0397	0.9606
R3190_p165_tmp_3090.dat	0.2142	0.0498	0.9504
R3090_r10_tmp_3195.dat	0.2784	0.0391	0.9611
R3197tr_tmp_3066.dat	0.3102	0.0387	0.9615
Test Class 1			
R3042_tmp_3042.dat	0.0682	0.9631	0.0364
R3051_tmp_3051.dat	0.0942	0.9631	0.0364
R3090_tmp_3090.dat	0.1734*	0.9631	0.0363
R3190_p165_tmp_3190_p165.dat	0.0437	0.9624	0.0371
R3190_p221_tmp_3190_p221.dat	0.0834	0.9623	0.0371
R3190_p47_tmp_3190_p47.dat	0.0514	0.9628	0.0366
R3195Tank_tmp_3195Tank.dat	0.1138	0.9579	0.0415
R3190_T227_tmp_3190_t227.dat	0.1214	0.9631	0.0363
R3033_tmp_3033.dat	0.4311	0.1164	0.0365
Test Class 2			
R3051_tmp_3195.dat	0.2707	0.0411	0.9592
R3195_r10_tmp_3028.dat	0.2355	0.0739	0.9264
R3195Tank_tmp_3028.dat	0.1784*	0.1866	0.8135
R3197tr_tmp_3028.dat	0.3136	0.0387	0.9615
R3197tr_tmp_3083.dat	0.3157	0.0386	0.9616
R3190_p221_tmp_3083.dat	0.2542	0.0425	0.9577
R3090_tmp_3083.dat	0.2673	0.0417	0.9585
R3074_tmp_3195.dat	0.2781	0.0394	0.9609
R3042_tmp_3195.dat	0.2915	0.0392	0.9610

* Represents a misclassified result or classification with a very low figure of merit.
(Note: With thresholds of 0.8 and 0.2, the neural net
gives perfect results with an excellent figure of merit.)

Table 6.12
Classification Results for Set 1 Binary Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 1			
R3028_btmp_3028.dat	0.1181	0.9701	0.0301
R3074_btmp_3074.dat	0.0376	0.9699	0.0303
R3066_btmp_3066.dat	0.1042	0.9701	0.0302
R3088_btmp_3088.dat	0.0540	0.9701	0.0302
R3083_btmp_3083.dat	0.0717	0.9696	0.0306
R3195_btmp_3195.dat	0.1079	0.9675	0.0328
R3197_btmp_3197.dat	0.0544	0.9699	0.0303
R3215_btmp_3215.dat	0.0942	0.9701	0.0301
R3197tr_btmp_3197tr.dat	0.1230	0.9440	0.0566
Training Class 2			
R3028_r10_btmp_3195.dat	0.2477	0.3699*	0.6312*
R3083_r10_btmp_3195.dat	0.2481	0.0460	0.9537
R3195_r10_btmp_3028.dat	0.1905	0.0479	0.9518
R3033_btmp_3028.dat	0.1728	0.0624	0.9372
R3190_p47_btmp_3090.dat	0.2468	0.0446	0.9551
R3033_btmp_3195.dat	0.2523	0.0443	0.9554
R3190_p165_btmp_3090.dat	0.1877	0.0610	0.9386
R3090_r10_btmp_3195.dat	0.2382	0.0450	0.9547
R3197tr_btmp_3066.dat	0.2757	0.0439	0.9558
Test Class 1			
R3042_btmp_3042.dat	0.0376	0.9701	0.0302
R3051_btmp_3051.dat	0.0640	0.9701	0.0302
R3090_btmp_3090.dat	0.0971	0.9701	0.0301
R3190_p165_btmp_3190_p165.dat	0.0326	0.9700	0.0303
R3190_p221_btmp_3190_p221.dat	0.0686	0.9699	0.0304
R3190_p47_btmp_3190_p47.dat	0.0539	0.9701	0.0302
R3195Tank_btmp_3195Tank.dat	0.1058	0.9681	0.0322
R3190_t227_btmp_3190_t227.dat	0.0578	0.9701	0.0302
R3033_btmp_3033.dat	0.0735	0.9695	0.0307
Test Class 2			
R3051_btmp_3195.dat	0.2632	0.0443	0.9554
R3195_r10_btmp_3028.dat	0.1905	0.0479	0.9518
R3195Tank_btmp_3028.dat	0.2548	0.0451	0.9546
R3197tr_btmp_3028.dat	0.2791	0.0439	0.9558
R3197tr_btmp_3083.dat	0.2794	0.0439	0.9558
R3190_p221_btmp_3083.dat	0.2273	0.0456	0.9540
R3090_btmp_3083.dat	0.2298	0.0455	0.9542
R3074_btmp_3195.dat	0.2528	0.0449	0.9548
R3042_btmp_3195.dat	0.2545	0.0447	0.9550

* Values progressed to 0.1655 and 0.8347 after 9000 training file iterations.
(Note: Both methods gave perfect classification.
No significant difference between range and binary data.)

Table 6.13
Classification Results for Set 2 Range Training Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 1			
R3028_tmp_3028.dat	0.1321	0.9842	0.0140
R3028_r40_tmp_3028.dat	0.0575	0.9836	0.0150
R3066_tmp_3066.dat	0.1390	0.9844	0.0137
R3066_r40_tmp_3066.dat	0.1342	0.9807	0.0184
R3074_tmp_3074.dat	0.0464	0.9840	0.0145
R3074_r40_tmp_3074.dat	0.1074	0.9786	0.0209
R3088_tmp_3088.dat	0.1272	0.9840	0.0144
R3088_r40_tmp_3088.dat	0.0419	0.9822	0.0169
R3083_tmp_3083.dat	0.0678	0.9835	0.0151
R3083_r40_tmp_3083.dat	0.0667	0.9824	0.0165
R3195_tmp_3195.dat	0.0906	0.9795	0.0200
R3195_r40_tmp_3195.dat	0.1207	0.9553	0.0465
R3042_tmp_3042.dat	0.1222	0.9844	0.0138
R3042_r40_tmp_3042.dat	0.1419	0.9730	0.0278
Training Class 2			
R3028_R10_tmp_3195.dat	0.2158	0.0177	0.9835
R3066_tmp_3195.dat	0.2394	0.0156	0.9862
R3066_tmp_3083.dat	0.2489	0.0154	0.9865
R3074_tmp_3195.dat	0.2221	0.0168	0.9846
R3074_tmp_3090.dat	0.1734	0.0220	0.9794
R3088_tmp_3190_p47.dat	0.2160	0.0170	0.9843
R3088_tmp_3083.dat	0.2119	0.0191	0.9821
R3083_r10_tmp_3195.dat	0.2213	0.0173	0.9841
R3083_r10_tmp_3028.dat	0.2057	0.0204	0.9808
R3195_r10_tmp_3028.dat	0.1828	0.0421	0.9576
R3042_tmp_3195.dat	0.2366	0.0164	0.9851
R3042_tmp_3090.dat	0.2093	0.0235	0.9768

(Note: Distance threshold at 0.16)

Table 6.14
Classification Results for Set 2 Range Test Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Test Class 1			
R3028_r10_tmp_3028.dat	0.0579	0.9834	0.0153
R3028_r20_tmp_3028.dat	0.0501	0.9837	0.0148
R3028_r30_tmp_3028.dat	0.0528	0.9835	0.0151
R3066_r10_tmp_3066.dat	0.1119	0.9824	0.0168
R3066_r20_tmp_3066.dat	0.1468	0.9825	0.0166
R3066_r30_tmp_3066.dat	0.0997	0.9832	0.0156
R3074_r10_tmp_3074.dat	0.1102	0.8776	0.1226
R3074_r20_tmp_3074.dat	0.1134	0.8258	0.1732
R3074_r30_tmp_3074.dat	0.1401	0.6706*	0.3310
R3088_r10_tmp_3088.dat	0.0455	0.9828	0.0161
R3088_r20_tmp_3088.dat	0.0355	0.9833	0.0154
R3088_r30_tmp_3088.dat	0.0397	0.9836	0.0150
R3083_r10_tmp_3083.dat	0.0723	0.9818	0.0172
R3083_r20_tmp_3083.dat	0.0699	0.9828	0.0160
R3083_r30_tmp_3083.dat	0.0655	0.9824	0.0165
R3195_r10_tmp_3195.dat	0.1291	0.9112	0.0917
R3195_r20_tmp_3195.dat	0.1264	0.9380	0.0647
R3195_r30_tmp_3195.dat	0.1174	0.9489	0.0528
R3042_r10_tmp_3042.dat	0.1162	0.9801	0.0191
R3042_r20_tmp_3042.dat	0.0929	0.9822	0.0167
R3042_r30_tmp_3042.dat	0.1419	0.9808	0.0186
Test Class 2			
R3028_r40_tmp_3195.dat	0.2489	0.0159	0.9858
R3066_r40_tmp_3195.dat	0.2410	0.0159	0.9858
R3066_r20_tmp_3195.dat	0.2359	0.0159	0.9858
R3066_r40_tmp_3083.dat	0.2482	0.0153	0.9867
R3066_r20_tmp_3083.dat	0.2026	0.0191	0.9819
R3074_r40_tmp_3195.dat	0.2454	0.0154	0.9866
R3074_r20_tmp_3195.dat	0.2285	0.0161	0.9856
R3074_r40_tmp_3090.dat	0.2010	0.0360	0.9623
R3074_r20_tmp_3090.dat	0.1677	0.0416	0.9575
R3088_r20_tmp_3195.dat	0.2402	0.0159	0.9858
R3088_r40_tmp_3195.dat	0.2468	0.0167	0.9849
R3088_r20_tmp_3083.dat	0.1887	0.0558	0.9417
R3088_r40_tmp_3083.dat	0.1732	0.1201	0.8722
R3042_r20_tmp_3090.dat	0.2208	0.0187	0.9834
R3042_r40_tmp_3090.dat	0.2286	0.0166	0.9851
R3042_r20_tmp_3195.dat	0.2433	0.0161	0.9854
R3042_r40_tmp_3195.dat	0.2362	0.0165	0.9850
R3083_r20_tmp_3195.dat	0.2262	0.0183	0.9828
R3083_r40_tmp_3195.dat	0.2250	0.0176	0.9837
R3083_r20_tmp_3028.dat	0.2153	0.4513	0.5635*
R3083_r40_tmp_3028.dat	0.1872	0.2668	0.7398*
R3195_r40_tmp_3028.dat	0.2280	0.0182	0.9833

* Represents a *non*classification with thresholds of 0.8 and 0.2
(Note: Distance classifications are good, but at the expense of a figure of merit.)

Table 6.15
Classification Results for Set 2 Binary Training Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 1			
R3028_btmp_3028.dat	0.1619*	0.9764	0.0213
R3028_r40_btmp_3028.dat	0.0615	0.9758	0.0231
R3066_btmp_3066.dat	0.1209	0.9766	0.0207
R3066_r40_btmp_3066.dat	0.1070	0.9677	0.0314
R3074_btmp_3074.dat	0.0345	0.9758	0.0226
R3074_r40_btmp_3074.dat	0.1030	0.9567	0.0438
R3088_btmp_3088.dat	0.0886	0.9757	0.0231
R3088_r40_btmp_3088.dat	0.0309	0.9724	0.0269
R3083_btmp_3083.dat	0.0582	0.9750	0.0235
R3083_r40_btmp_3083.dat	0.0663	0.9717	0.0274
R3195_btmp_3195.dat	0.0824	0.9713	0.0279
R3195_r40_btmp_3195.dat	0.1030	0.9569	0.0444
R3042_btmp_3042.dat	0.0772	0.9763	0.0216
R3042_r40_btmp_3042.dat	0.0980	0.9494	0.0523
Training Class 2			
R3028_R10_btmp_3195.dat	0.2066	0.0222	0.9812
R3066_btmp_3195.dat	0.2171	0.0222	0.9814
R3066_btmp_3083.dat	0.2159	0.0417	0.9639
R3074_btmp_3195.dat	0.2137	0.0221	0.9819
R3074_btmp_3090.dat	0.1521	0.0460	0.9557
R3088_btmp_3190_p47.dat	0.2002	0.0230	0.9806
R3088_btmp_3083.dat	0.1822	0.0346	0.9663
R3083_r10_btmp_3195.dat	0.2083	0.0234	0.9797
R3083_r10_btmp_3028.dat	0.1851	0.0243	0.9783
R3195_r10_btmp_3028.dat	0.1522	0.0439	0.9579
R3042_btmp_3195.dat	0.2138	0.0230	0.9804
R3042_btmp_3090.dat	0.1892	0.0310	0.9713

* Indicates a misclassification with a distance threshold of 0.14
(Note: the training set with the neural net is perfectly classified.)

Table 6.16
Classification Results for Set 2 Binary Test Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Test Class 1			
R3028_r10_btmp_3028.dat	0.0475	0.9758	0.0230
R3028_r20_btmp_3028.dat	0.0589	0.9759	0.0226
R3028_r30_btmp_3028.dat	0.0550	0.9754	0.0237
R3066_r10_btmp_3066.dat	0.1105	0.9570	0.0464
R3066_r20_btmp_3066.dat	0.1305	0.9729	0.0277
R3066_r30_btmp_3066.dat	0.1072	0.9758	0.0223
R3074_r10_btmp_3074.dat	0.1218	0.3602	0.6323*
R3074_r20_btmp_3074.dat	0.1076	0.7895	0.2161
R3074_r30_btmp_3074.dat	0.1263	0.5568	0.4477
R3088_r10_btmp_3088.dat	0.0313	0.9723	0.0274
R3088_r20_btmp_3088.dat	0.0254	0.9738	0.0259
R3088_r30_btmp_3088.dat	0.0206	0.9744	0.0246
R3083_r10_btmp_3083.dat	0.0716	0.9674	0.0325
R3083_r20_btmp_3083.dat	0.0664	0.9708	0.0286
R3083_r30_btmp_3083.dat	0.0666	0.9717	0.0271
R3195_r10_btmp_3195.dat	0.1111	0.9147	0.0903
R3195_r20_btmp_3195.dat	0.1116	0.9389	0.0648
R3195_r30_btmp_3195.dat	0.1019	0.9457	0.0566
R3042_r10_btmp_3042.dat	0.1012	0.7934*	0.2253
R3042_r20_btmp_3042.dat	0.0841	0.9701	0.0302
R3042_r30_btmp_3042.dat	0.0916	0.9493	0.0521
Test Class 2			
R3028_r40_btmp_3195.dat	0.2192	0.0220	0.9809
R3066_r40_btmp_3195.dat	0.2255	0.0218	0.9820
R3066_r20_btmp_3195.dat	0.2218	0.0222	0.9814
R3066_r40_btmp_3083.dat	0.2088	0.0260	0.9781
R3066_r20_btmp_3083.dat	0.1847	0.0256	0.9768
R3074_r40_btmp_3195.dat	0.2158	0.0221	0.9809
R3074_r20_btmp_3195.dat	0.2018	0.0228	0.9806
R3074_r40_btmp_3090.dat	0.1941	0.0243	0.9780
R3074_r20_btmp_3090.dat	0.1937	0.0237	0.9793
R3088_r20_btmp_3195.dat	0.2188	0.0217	0.9819
R3088_r40_btmp_3195.dat	0.2138	0.0221	0.9814
R3088_r20_btmp_3083.dat	0.1803	0.0356	0.9649
R3088_r40_btmp_3083.dat	0.1637	0.0580	0.9426
R3042_r20_btmp_3090.dat	0.1669	0.0254	0.9777
R3042_r40_btmp_3090.dat	0.1680	0.0353	0.9660
R3042_r20_btmp_3195.dat	0.2163	0.0217	0.9822
R3042_r40_btmp_3195.dat	0.2089	0.0231	0.9808
R3083_r20_btmp_3195.dat	0.2179	0.0232	0.9799
R3083_r40_btmp_3195.dat	0.2154	0.0231	0.9800
R3083_r20_btmp_3028.dat	0.1947	0.0239	0.9786
R3083_r40_btmp_3028.dat	0.1890	0.0280	0.9732
R3195_r40_btmp_3028.dat	0.1772	0.0266	0.9749

* Represents a *non*classification with thresholds of 0.8 and 0.2
(Note: Both methods give good classification results.)

Table 6.17
Classification Results for Class 1 Set 3 Range Training Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 1			
R3197tr_tmp_3195.dat	0.1548*	0.0973	0.9027**
R3190_p165_tmp_3195.dat	0.1565*	0.9799	0.0193
R3190_p221_tmp_3195.dat	0.1748*	0.0973	0.9028**
R3074_tmp_3083.dat	0.1222	0.9837	0.0156
R3051_tmp_3090.dat	0.1268	0.9764	0.0228
R3042_tmp_3083.dat	0.1117	0.9988	0.0012
R3090_r40_tmp_3028.dat	0.1822*	0.9970	0.0030
R3090_tmp_3028.dat	0.1602*	0.9956	0.0043
R3033_r40_tmp_3083.dat	0.1444*	0.9837	0.0156
R3033_r10_tmp_3083.dat	0.1278	0.9837	0.0156
R3033_tmp_3083.dat	0.1321	0.9837	0.0156
R3028_tmp_3028.dat	0.2069*	0.9970	0.0030
R3028_r40_tmp_3028.dat	0.0887	0.9970	0.0030
R3066_tmp_3066.dat	0.2057*	0.9970	0.0030
R3066_r40_tmp_3066.dat	0.1121	0.9968	0.0032
R3074_tmp_3074.dat	0.1187	0.9970	0.0030
R3074_r40_tmp_3074.dat	0.0625	0.9963	0.0037
R3088_tmp_3088.dat	0.1978*	0.9970	0.0030
R3088_r40_tmp_3088.dat	0.0842	0.9970	0.0030
R3083_tmp_3083.dat	0.1157	0.9970	0.0030
R3083_r40_tmp_3083.dat	0.0605	0.9970	0.0030
R3195_tmp_3195.dat	0.0861	0.9970	0.0030
R3195_r40_tmp_3195.dat	0.0679	0.9969	0.0031
R3042_tmp_3042.dat	0.2021*	0.9970	0.0030
R3042_r40_tmp_3042.dat	0.1114	0.9970	0.0030

* Represents a misclassification at a distance threshold of 0.135
** Represents a neural net misclassification
(Note: The two files that could not be trained with the neural net have a large scale difference between template and target.)

Table 6.18
Classification Results for Class 2 Set 3 Range Training Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 2			
R3197TR_tmp_3028.dat	0.1784	0.0975	0.9025
R3197TR_tmp_3083.dat	0.1803	0.0972	0.9028
R3190_p221_tmp_3090.dat	0.1048*	0.0976	0.9024
R3190_p221_tmp_3083.dat	0.1200*	0.0974	0.9027
R3190_p165_tmp_3090.dat	0.1062*	0.0977	0.9024
R3190_p165_tmp_3083.dat	0.1249*	0.0973	0.9027
R3051_tmp_3195.dat	0.1366	0.0973	0.9027
R3051_tmp_3083.dat	0.1599	0.0973	0.9027
R3090_tmp_3083.dat	0.1343*	0.0974	0.9026
R3090_tmp_3195.dat	0.1411*	0.0973	0.9027
R3033_tmp_3028.dat	0.1241*	0.0974	0.9027
R3033_tmp_3195.dat	0.1566	0.0974	0.9027
R3028_r10_tmp_3195.dat	0.1365	0.0973	0.9027
R3066_tmp_3195.dat	0.1596	0.0974	0.9027
R3066_tmp_3083.dat	0.1688	0.0972	0.9028
R3074_tmp_3195.dat	0.1413	0.0985	0.9015
R3074_tmp_3090.dat	0.1060*	0.0975	0.9025
R3088_tmp_3195.dat	0.1597	0.0973	0.9027
R3088_tmp_3083.dat	0.1383	0.1000	0.9000
R3083_r10_tmp_3195.dat	0.1427	0.0974	0.9027
R3083_r10_tmp_3028.dat	0.1439	0.0979	0.9022
R3195_r10_tmp_3028.dat	0.1188*	0.0974	0.9027
R3042_tmp_3195.dat	0.1586	0.0972	0.9028
R3042_tmp_3090.dat	0.1535	0.0974	0.9027

* Represents a misclassification at a threshold of 0.135
(Note: The ability of a neural net to train and seperate very poorly distributed data is outstanding. The key to classifying test data, is to insure the training data is a good representation of *all* the data.)

Table 6.19
Classification Results for Class 1 Set 3 Range Test Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Test Class 1			
R3190_tmp_3083.dat	0.1441*	0.0974	0.9026*
R3088_tmp_3090.dat	0.1372*	0.9837	0.0155
R3035_tmp_3083.dat	0.1256	0.9829	0.0164
R3066_tmp_3090.dat	0.1088	0.0972	0.9028*
R3190_p47_tmp_3195.dat	0.1629*	0.1032	0.8966*
R3197TR_r40_tmp_3195.dat	0.1582*	0.0973	0.9027*
R3195TANK_tmp_3083.dat	0.1668*	0.9770	0.0220
R3195TANK_r40_tmp_3083.dat	0.1632*	0.0992	0.9008*
R3090_r30_tmp_3028.dat	0.1443*	0.6513**	0.3472
R3090_r20_tmp_3028.dat	0.1327	0.0977	0.9024*
R3090_r10_tmp_3028.dat	0.1780*	0.0974	0.9026*
R3033_r30_tmp_3083.dat	0.1374*	0.9837	0.0156
R3033_r20_tmp_3083.dat	0.1288	0.9837	0.0156
R3028_r10_tmp_3028.dat	0.0829	0.9970	0.0030
R3028_r20_tmp_3028.dat	0.0901	0.9970	0.0030
R3028_r30_tmp_3028.dat	0.1026	0.9970	0.0030
R3066_r10_tmp_3066.dat	0.1092	0.9970	0.0030
R3066_r20_tmp_3066.dat	0.1428*	0.9970	0.0030
R3066_r30_tmp_3066.dat	0.0980	0.9970	0.0030
R3074_r10_tmp_3074.dat	0.0451	0.9169	0.0823
R3074_r20_tmp_3074.dat	0.0541	0.9963	0.0036
R3074_r30_tmp_3074.dat	0.0716	0.9851	0.0147
R3088_r10_tmp_3088.dat	0.1014	0.9970	0.0030
R3088_r20_tmp_3088.dat	0.0942	0.9970	0.0030
R3088_r30_tmp_3088.dat	0.1059	0.9970	0.0030
R3083_r10_tmp_3083.dat	0.0580	0.9970	0.0030
R3083_r20_tmp_3083.dat	0.0609	0.9970	0.0030
R3083_r30_tmp_3083.dat	0.0614	0.9970	0.0030
R3195_r10_tmp_3195.dat	0.0689	0.9924	0.0075
R3195_r20_tmp_3195.dat	0.0688	0.9967	0.0032
R3195_r30_tmp_3195.dat	0.0664	0.9969	0.0030
R3042_r10_tmp_3042.dat	0.0956	0.9970	0.0030
R3042_r20_tmp_3042.dat	0.0774	0.9970	0.0030
R3042_r30_tmp_3042.dat	0.1150	0.9969	0.0030

* Represents a misclassification
** Represents a *non*classification
(Note: All the neural net misclassifications are in the same target-different view autocorrelation class. See note below Table 6.18.)

Table 6.20
Classification Results for Class 2 Set 3 Range Test Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Test Class 2			
R3195TANK_tmp_3028.dat	0.0866*	0.9973*	0.0027
R3195TANK_tmp_3195.dat	0.1376	0.0974	0.9027
R3197TR_tmp_3090.dat	0.1771	0.0973	0.9028
R3197TR_tmp_3083.dat	0.1803	0.0972	0.9028
R3190_p47_tmp_3090.dat	0.1067*	0.1071	0.8929
R3190_p47_tmp_3083.dat	0.1028*	0.0974	0.9027
R3190_T227_tmp_3090.dat	0.0825*	0.0976	0.9025
R3190_T227_tmp_3195.dat	0.1385	0.0973	0.9027
R3035_tmp_3090.dat	0.0927*	0.0974	0.9026
R3035_tmp_3195.dat	0.1386	0.0973	0.9027
R3090_r10_tmp_3083.dat	0.1278*	0.0974	0.9027
R3090_r10_tmp_3195.dat	0.1411	0.1005	0.8994
R3090_r20_tmp_3083.dat	0.1206*	0.9836*	0.0157
R3090_r30_tmp_3083.dat	0.1247*	0.1542	0.8443
R3090_r20_tmp_3195.dat	0.1420	0.1011	0.8988
R3090_r30_tmp_3195.dat	0.1430	0.9837*	0.0156
R3033_r20_tmp_3028.dat	0.1413	0.9837*	0.0156
R3033_r30_tmp_3028.dat	0.1020*	0.0975	0.9026
R3033_r20_tmp_3195.dat	0.1618	0.9795*	0.0197
R3033_r30_tmp_3195.dat	0.1491	0.0972	0.9028
R3028_r40_tmp_3195.dat	0.1678	0.0975	0.9025
R3066_r40_tmp_3195.dat	0.1597	0.9558*	0.0423
R3066_r20_tmp_3195.dat	0.1553	0.0975	0.9025
R3066_r40_tmp_3083.dat	0.1674	0.0977	0.9023
R3066_r20_tmp_3083.dat	0.1228*	0.1699	0.8284
R3074_r40_tmp_3195.dat	0.1649	0.0973	0.9027
R3074_r20_tmp_3195.dat	0.1485	0.0973	0.9027
R3074_r40_tmp_3090.dat	0.1320*	0.0974	0.9027
R3074_r20_tmp_3090.dat	0.0916*	0.0974	0.9027
R3088_r20_tmp_3195.dat	0.1596	0.9837*	0.0156
R3088_r40_tmp_3195.dat	0.1685	0.9640*	0.0346
R3088_r20_tmp_3083.dat	0.1150*	0.0978	0.9022
R3088_r40_tmp_3083.dat	0.1031*	0.0974	0.9027
R3042_r20_tmp_3090.dat	0.1753	0.0974	0.9027
R3042_r40_tmp_3090.dat	0.1601	0.0973	0.9027
R3042_r20_tmp_3195.dat	0.1646	0.0972	0.9028
R3042_r40_tmp_3195.dat	0.1569	0.1091	0.8907
R3083_r20_tmp_3195.dat	0.1481	0.0975	0.9026
R3083_r40_tmp_3195.dat	0.1473	0.0974	0.9027
R3083_r20_tmp_3028.dat	0.1858	0.9994*	0.0006
R3083_r40_tmp_3028.dat	0.1414	0.0995	0.9005
R3195_r40_tmp_3028.dat	0.1581	0.0976	0.9025

* Represents a misclassification

Table 6.21
Classification Results for Class 1 Set 3 Binary Training Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 1			
R3197tr_btmp_3195.dat	0.1421*	0.0885	0.9113**
R3190_p165_btmp_3195.dat	0.1540*	0.9998	0.0002
R3190_p221_btmp_3195.dat	0.1503*	0.0887	0.9111**
R3074_btmp_3083.dat	0.1229	0.9997	0.0002
R3051_btmp_3090.dat	0.0955	0.9964	0.0029
R3042_btmp_3083.dat	0.0927	0.9944	0.0047
R3090_r40_btmp_3028.dat	0.1132	0.9888	0.0113
R3090_btmp_3028.dat	0.0948	0.9959	0.0034
R3033_r40_btmp_3083.dat	0.1345*	0.9964	0.0042
R3033_r10_btmp_3083.dat	0.1270*	0.9822	0.0198
R3033_btmp_3083.dat	0.1184	0.9721	0.0314
R3028_btmp_3028.dat	0.2383*	0.9941	0.0049
R3028_r40_btmp_3028.dat	0.1242	0.9939	0.0051
R3066_btmp_3066.dat	0.1831*	0.9939	0.0051
R3066_r40_btmp_3066.dat	0.0706	0.9938	0.0052
R3074_btmp_3074.dat	0.1028	0.9924	0.0061
R3074_r40_btmp_3074.dat	0.0453	0.9937	0.0052
R3088_btmp_3088.dat	0.1597*	0.9908	0.0081
R3088_r40_btmp_3088.dat	0.0696	0.9948	0.0044
R3083_btmp_3083.dat	0.1036	0.9995	0.0004
R3083_r40_btmp_3083.dat	0.0510	0.9966	0.0028
R3195_btmp_3195.dat	0.0938	0.9985	0.0012
R3195_r40_btmp_3195.dat	0.0668	0.9963	0.0030
R3042_btmp_3042.dat	0.1554*	0.9939	0.0051
R3042_r40_btmp_3042.dat	0.0635	0.9939	0.0051

* Represents a misclassification at a threshold of 0.125
** Represents a neural net misclassification
(Note: The two files that could not be trained with the neural net have a large scale difference between template and target. Also, these are the same two files that could not be trained using the range data.)

Table 6.22
Classification Results for Class 2 Set 3 Binary Training Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Training Class 2			
R3197TR_btmp_3028.dat	0.1616	0.0887	0.9111
R3197TR_btmp_3083.dat	0.1624	0.0887	0.9111
R3190_p221_btmp_3090.dat	0.1326	0.0777	0.9223
R3190_p221_btmp_3083.dat	0.1117*	0.0866	0.9133
R3190_p165_btmp_3090.dat	0.0827*	0.0887	0.9111
R3190_p165_btmp_3083.dat	0.1254	0.0051	0.9952
R3051_btmp_3195.dat	0.1514	0.0885	0.9113
R3051_btmp_3083.dat	0.1289	0.0887	0.9111
R3090_btmp_3083.dat	0.1106*	0.0879	0.9119
R3090_btmp_3195.dat	0.1268	0.0858	0.9141
R3033_btmp_3028.dat	0.0712*	0.0365	0.9613
R3033_btmp_3195.dat	0.1410	0.0892	0.9106
R3028_R10_btmp_3195.dat	0.1278	0.0964	0.9037
R3066_btmp_3195.dat	0.1398	0.0887	0.9111
R3066_btmp_3083.dat	0.1494	0.0892	0.9106
R3074_btmp_3195.dat	0.1398	0.0887	0.9111
R3074_btmp_3090.dat	0.0838*	0.0039	0.9963
R3088_btmp_3195.dat	0.1421	0.0886	0.9112
R3088_btmp_3083.dat	0.1097*	0.0428	0.9557
R3083_r10_btmp_3195.dat	0.1343	0.0887	0.9111
R3083_r10_btmp_3028.dat	0.1133*	0.0050	0.9952
R3195_r10_btmp_3028.dat	0.0942*	0.0039	0.9963
R3042_btmp_3195.dat	0.1352	0.0909	0.9091
R3042_btmp_3090.dat	0.1167*	0.0063	0.9940

* Represents a misclassification at a threshold of 0.125
(Note: The ability of a neural net to train and seperate very poorly distributed data is outstanding. The key to classifying test data, is to insure the training data is a good representation of *all* the data. Same baisc results as with the range data.)

Table 6.23
Classification Results for Class 1 Set 3 Binary Test Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Test Class 1			
R3190_btmp_3083.dat	0.1326*	0.0884	0.9114*
R3088_btmp_3090.dat	0.1392*	0.0889	0.9109*
R3035_btmp_3083.dat	0.1190	0.9994	0.0005
R3066_btmp_3090.dat	0.0774	0.9983	0.0014
R3190_p47_btmp_3195.dat	0.1559*	0.0886	0.9112*
R3197TR_r40_btmp_3195.dat	0.1465*	0.0887	0.9111*
R3195TANK_btmp_3083.dat	0.1013	0.9441	0.0617
R3195TANK_r40_btmp_3083.dat	0.1441*	0.9998	0.0002
R3090_r30_btmp_3028.dat	0.1093	0.9951	0.0041
R3090_r20_btmp_3028.dat	0.1001	0.8536	0.1612
R3090_r10_btmp_3028.dat	0.0897	0.9981	0.0016
R3033_r30_btmp_3083.dat	0.1247	0.0813	0.9187*
R3033_r20_btmp_3083.dat	0.1287*	0.0990	0.9013*
R3028_r10_btmp_3028.dat	0.1095	0.9939	0.0051
R3028_r20_btmp_3028.dat	0.1174	0.9916	0.0067
R3028_r30_btmp_3028.dat	0.0987	0.9927	0.0059
R3066_r10_btmp_3066.dat	0.0635	0.9937	0.0052
R3066_r20_btmp_3066.dat	0.1049	0.9939	0.0051
R3066_r30_btmp_3066.dat	0.1076	0.9939	0.0051
R3074_r10_btmp_3074.dat	0.0542	0.0192	0.9813*
R3074_r20_btmp_3074.dat	0.0387	0.9911	0.0074
R3074_r30_btmp_3074.dat	0.0540	0.9763	0.0216
R3088_r10_btmp_3088.dat	0.0777	0.5200**	0.4432
R3088_r20_btmp_3088.dat	0.0748	0.9939	0.0051
R3088_r30_btmp_3088.dat	0.0769	0.9939	0.0051
R3083_r10_btmp_3083.dat	0.0484	0.9975	0.0021
R3083_r20_btmp_3083.dat	0.0485	0.9955	0.0037
R3083_r30_btmp_3083.dat	0.0504	0.9954	0.0038
R3195_r10_btmp_3195.dat	0.0641	0.9971	0.0024
R3195_r20_btmp_3195.dat	0.0667	0.9978	0.0018
R3195_r30_btmp_3195.dat	0.0658	0.9984	0.0013
R3042_r10_btmp_3042.dat	0.0553	0.9730	0.0240
R3042_r20_btmp_3042.dat	0.0629	0.9902	0.0077
R3042_r30_btmp_3042.dat	0.0551	0.9933	0.0055

* Represents a misclassification
** Represents a *non*classification
(Note: All the neural net misclassifications are in the same target-different view autocorrelation class. See note below Table 6.22. Same basic results as with the range data.)

Table 6.24
Classification Results for Class 2 Set 3 Binary Test Data
(Note: Distances are calculated from the center of the class 1 training data.)

File Name	Distance	Neural Net	
		Node 1	Node 2
Test Class 2			
R3195TANK_btmp_3028.dat	0.1557	0.0887	0.9111
R3195TANK_btmp_3195.dat	0.1213*	0.0076	0.9926
R3197TR_btmp_3090.dat	0.1632	0.0887	0.9111
R3197TR_btmp_3083.dat	0.1624	0.0887	0.9111
R3190_p47_btmp_3090.dat	0.1281	0.8735*	0.1453
R3190_p47_btmp_3083.dat	0.0826*	0.9697*	0.0290
R3190_T227_btmp_3090.dat	0.0984*	0.0121	0.9882
R3190_T227_btmp_3195.dat	0.1449	0.0886	0.9112
R3035_btmp_3090.dat	0.1370	0.0039	0.9963
R3035_btmp_3195.dat	0.1320	0.0174	0.9831
R3090_r10_btmp_3083.dat	0.1294	0.9850*	0.0192
R3090_r10_btmp_3195.dat	0.1188*	0.1523	0.8483
R3090_r20_btmp_3083.dat	0.1215*	0.0882	0.9116
R3090_r30_btmp_3083.dat	0.1072*	0.1105	0.8900
R3090_r20_btmp_3195.dat	0.1241*	0.0988	0.9014
R3090_r30_btmp_3195.dat	0.1169*	0.0883	0.9115
R3033_r20_btmp_3028.dat	0.1545	0.0039	0.9963
R3033_r30_btmp_3028.dat	0.0863*	0.9735*	0.0221
R3033_r20_btmp_3195.dat	0.1323	0.0883	0.9115
R3033_r30_btmp_3195.dat	0.1528	0.0887	0.9111
R3028_r40_btmp_3195.dat	0.1440	0.0887	0.9111
R3066_r40_btmp_3195.dat	0.1498	0.0887	0.9111
R3066_r20_btmp_3195.dat	0.1472	0.0887	0.9111
R3066_r40_btmp_3083.dat	0.1356	0.0887	0.9111
R3066_r20_btmp_3083.dat	0.1060*	0.9968*	0.0037
R3074_r40_btmp_3195.dat	0.1426	0.0887	0.9111
R3074_r20_btmp_3195.dat	0.1228*	0.1061	0.8943
R3074_r40_btmp_3090.dat	0.1202*	0.0042	0.9960
R3074_r20_btmp_3090.dat	0.1245*	0.0887	0.9111
R3088_r20_btmp_3195.dat	0.1424	0.0887	0.9111
R3088_r40_btmp_3195.dat	0.1406	0.0886	0.9112
R3088_r20_btmp_3083.dat	0.1071*	0.9790*	0.0213
R3088_r40_btmp_3083.dat	0.0912*	0.9847*	0.0145
R3042_r20_btmp_3090.dat	0.0934*	0.0880	0.9118
R3042_r40_btmp_3090.dat	0.0914*	0.0131	0.9869
R3042_r20_btmp_3195.dat	0.1378	0.0877	0.9121
R3042_r40_btmp_3195.dat	0.1336	0.0883	0.9115
R3083_r20_btmp_3195.dat	0.1421	0.0909	0.9091
R3083_r40_btmp_3195.dat	0.1423	0.0887	0.9111
R3083_r20_btmp_3028.dat	0.1216*	0.0887	0.9111
R3083_r40_btmp_3028.dat	0.1175*	0.0885	0.9113
R3195_r40_btmp_3028.dat	0.1067*	0.4113	0.6000**

* Represents a misclassification
** Represents a *non*classification

VII. Conclusions and Recommendations

7.1 Conclusions

This thesis examined methods of classifying and locating both segmented and non-segmented targets using laser range and binary data. The classification methods included applying both standard distance measurements and a neural network to the peak of a PSRI space correlation. The experimental results on these methods demonstrated that a trainable neural network has distinct advantages over distance measurements, both in absolute classification rates and in the figure of merit area. The study with the multilayer perceptron also demonstrated that these are not magical devices. Care must be taken to provide the network with an adequate representation of the data if satisfactory classification results are to be obtained. In a real world situation, many more training files would be needed to ensure proper classification. However, the successful results demonstrated in this thesis indicate that this may be a classification alternative worth pursuing.

This study indicates that with the chosen features and methods of classification, the relative range information did not provide any extra useful information about the target. Again, as stated in chapter VI, this does not mean that the information isn't available, only that the chosen features and classification method didn't make use of this information. It could also mean that the extra information wasn't required. The classification rates in sets one and two using the neural network were near 100% for both binary and range data. Extra information can't be deemed useful if it isn't needed. In set three, the classification rate was much less, particularly with the same class - different view autocorrelations. However, this was most likely due to the limited number of training files of this type used. The success achieved with the set three training files gives hope that a larger training set could yield better classification results.

The other area explored was that of the space domain Goodman - Schwarz correlation. The theory of this correlation indicated that through a process of local normaliza-

tion, the peak in the correlation plane will correspond to where the input scene best numerically matched the template.

The test case for this experiment worked well with the peak in the correlation plane properly locating the target with two of the three templates tested. It was determined that for this test case, a threshold of 98% of the maximum could be used as a threshold for the purpose of classification. However, a more extensive study is needed to determine some type of universal threshold. A multistage process was used to greatly reduce the computation time required. This multistage process also allowed for locating partially occluded targets. This process was found to work well on the larger targets but not on the small target. Again, more study is needed to determine size limitations. The Goodman - Schwarz correlation could be set up in a parallel architecture just as the basic correlation could, but, it would be a massive undertaking. The highlight of the Goodman - Schwarz correlation is that it works through a local energy normalization operation. Local normalization is essential in locating a target with this type of input data where large amounts of noise are distributed throughout the scene.

7.2 Recommendations

Many possibilities exist for further study. A larger number of correlations could be obtained to further test the application with neural networks. A way to create scaled versions of range images would be useful for creating training files. A determination needs to be made as to how scale and rotation invariant this algorithm is with respect to classification. This researcher strongly believes that the PSRI space is much more rotation invariant than scale invariant with respect to classification. However, this thesis is certainly not presented as a proof of this statement. A frequency filter was used when mapping the magnitude Fourier transform to the ($Ln\ r$, θ) coordinate system. More study is needed to determine an optimum filter. As a side note, if many more PSRI space correlations are to be performed using the Kobel and Martin Executive program [1:Vol II], one of the first items that should be accomplished is to modify the way the

data is stored. The present program requires too much memory. Certainly one possibility is too extract the peak without saving the correlation plane.

The files used with the neural network contained raw data taken from the correlation peaks. During the network operations, each file was normalized prior to processing by the net. In the training phase, a single file is then read and normalized each time it is used. This is extremely inefficient and takes approximately 30 minutes of cpu time, on a Micro Vax, for 1000 training file iterations using 100 nodes in the first hidden layer and 3 nodes in the second hidden layer. Once the desired form the data is known, files should be created that contain only this data before being processed by the neural network. This thesis only explored one of a large number of neural networks. Other types may be explored and may be found to yield superior results.

Mike Mayo, a fellow AFIT student, has demonstrated the optical transformation of a template into the PSRI feature space and the subsequent correlation of the PSRI spaces [16]. Therefore, a very exciting problem that could be immediately attacked would be to implement a hybrid optical/digital electronic system that could perform classification with a neural network using these correlation peaks. Because of time constraints, the optical transformation is essential in a real world application.

Appendix A. Neural Networks

The recent rediscovery of neural networks is due to new net topologies and algorithms. Also, the extensive calculations required for speech and pattern recognition have a great advantage in the parallel architecture of neural networks. The neural network used in this study was a multilayer perceptron which makes use of a backward error propagation routine. A complete description of this network can be found in a very good tutorial format article by Lippmann [22]. Conceptually, a multilayer perceptron is constructed as in fig. A.1. The input to the network is the set of features that are to be used for classification and the output becomes the class in which the features are grouped. Connection weights and node thresholds are initially chosen to be small random numbers and are updated during the training phase using the backward error propagation routine.

During the training phase, the net processes many examples of the classes that are desired to be classified. The weights and thresholds of the net are forming decision regions in an N dimensional space where N is the number of input features. The weights and internal thresholds are updated with the backward error propagation routine that updates values based on the actual values as compared to the desired output values. The backward error propagation routine makes the decision regions better accommodate the input data. Once the training is complete, the hope is that new data from the classes will also fall into the proper decision region, therefore being classified. This is very simplified and much more detail can be found in Lippmann [22]. There are, however, a few items that need to be stressed from or added to Lippmann.

First, the selection of the number of internal nodes. Lippmann states that in the second hidden layer, each node will identify a decision region [22:16]. Therefore, if there are only two decision regions, the output node can do the identification and there is no need for the second hidden layer. If the net is presented with a 2 class problem and each class is separated into 3 regions, then there needs to be at least 6 nodes in the

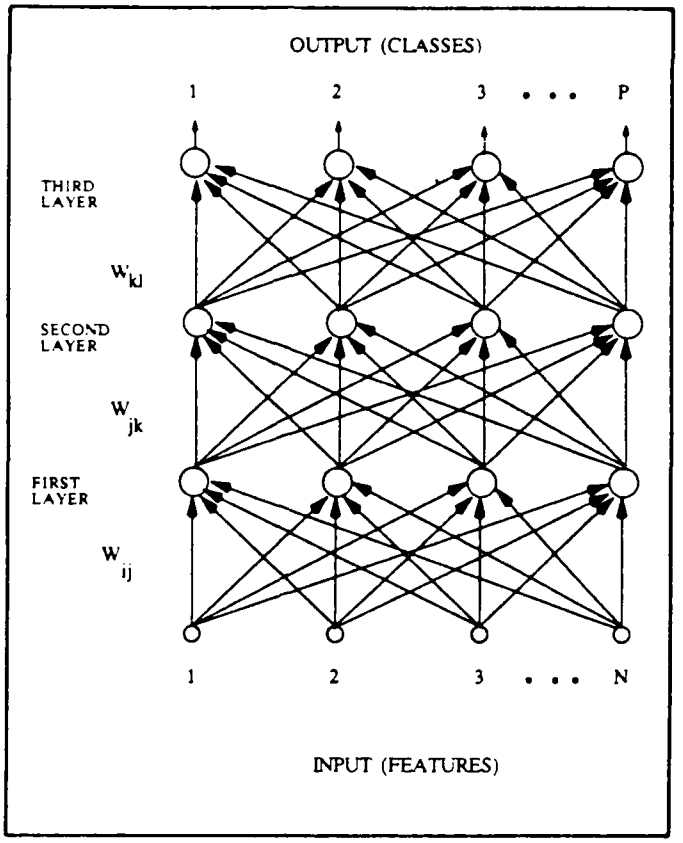


Fig. A.1 Conceptual Diagram of a Multilayer Perceptron

second hidden layer. Don't make the "the more nodes the better" mistake. At most, there need only be the same amount of regions as training examples. For the first hidden layer, Lippmann states that one node is needed for every boundary plane of each decision region. The boundary planes he is speaking of are really N-1 dimensional units. He therefore states the need for at least three times as many nodes in the first hidden layer as in the second hidden layer [22:16]. This is true if the data has been normalized so as to fall onto the surface of a hypersphere. If the data occupies the volume of the hypersphere, and it's necessary to completely enclose the decision region, there needs to be at least N+1 number of nodes for every region. For the case where there is no insight as to the distribution of the decision region, the following sequence is suggested:

1) Start training with two nodes in the second hidden layer and 2*(N+1) nodes in the first hidden layer.

2) Gradually increase the number of second hidden layer nodes (K) and first hidden layer nodes K*(N+1) until training is successful.

3) Hold the second hidden layer at the node number where successful training was achieved. Decrease the number of first hidden layer nodes to determine number required for separation.

This process was found to yield good groupings of the decision regions. It is possible that data could be grouped in a volume such that more than K*(N+1) nodes are required in the first hidden layer particularly when working in a lower dimensional feature space. In this case, step 2 will never be successful. Therefore, it may be necessary to find where step 2 was most successful and then start increasing the number of second hidden layer nodes.

It would be ideal if there was an infinite set of data to train the net with. The net could train on all possible views of a object and when training was successful, successful classification with any possible test data would be assured. With pattern recognition, the game is to train with a finite set of training data and then test with different data not included in the training set. The training sets are presented to the net in a random order so that the net can make as many training runs as needed. A problem with this is that the training set may not be a good representation of all the data. There have been attempts to quantify the required number of training sets per class needed to properly train the net, but this is very application dependent. For example, four training sets for a class would be enough if all the test data for that class fell within the region that would be bounded by the four training examples.

It can sometimes be hard to determine when to stop training and say that the net is not going to separate. The net is being trained to drive the correct output nodes to a value of one and drive the incorrect output nodes to zero. When a particular training file stabilizes at output values between 0.3 and 0.7, the chosen number of nodes is not going to separate the data. If the values are in this region but still flopping around a great deal, the net needs to train for a longer period. This also gets into a figure of merit question.

If the input data is classified based on the maximum output node, when can the decision of the net be trusted? In Lippmann's use of a multilayer perceptron with a sigmoidal type output rule, he used values of > 0.9 to indicate a definite true and values of < 0.1 to indicate a definite false [22:17]. This, of coarse, is fairly arbitrary and could be changed depending on the application.

Another factor that was critical to the successful net training was the selection of the initial weights and thresholds. At first these were all chosen from a random distribution centered at zero with a radius of 0.1. The resulting net failed miserably. When the radius was increased to 0.5, the net could successfully be trained. No exhaustive effort was made to logically explain this effect and it is offered only to possibly help someone else working in a similar area. It could have a sound mathematical basis or may be due to computer errors involved with many multiplications, additions, and subtractions with small numbers.

A final step is the preprocessing of the input data before sending it into the net. The inputs to the net were the 49 normalized components around the peak of the PSRI feature space correlation. Even though the resulting 49 dimensional hypersphere has an enormous surface area, the decision regions were greatly restricted due to using only positive inputs. With only positive inputs, the surface area is restricted to only one sector or $\frac{1}{2^{49}}$ of the possible area. Also, since all correlations have *generally* similar shapes around the peak, with a peak and a roll off from the peak, it's reasonable to assume that all the data would cluster into a relatively small area of this one sector. It appears that these groupings required too fine of a detail for the net to adequately separate. Therefore, the normalized input data was preprocessed to spread the data to fully occupy a unit hypercube. This greatly increases the separation between points and leads to much improved classification. The method is to first search all the components of the training data and find the minimum and maximum value for each component. The minimum value for each component is now set to zero and the maximum value is

set to one. The rest of the data is then spread in proportion to the original spread of the data [23]. This spreading operation did not yield better groupings with respect to distance measurements, but, it allowed for regions to be constructed using the neural network.

Appendix B. Computer Programs

The following is a listing of the computer programs that were written in support of this thesis effort. They include a complete listing for the multilayer perceptron, the peak extractor used to find and extract the peak of the PSRI space correlation, and a listing for the Goodman - Schwarz space domain correlation. These programs are all written in the VAX ADA programing language.

```
--              ********************************
--              * PROGRAM TITLE : MP           *
--              *                              *
--              * AUTHORS : LT STEVE TROXEL    *
--              *           CAPT DENNIS RUCK   *
--              ********************************

--   ******************************************************
--   * DISCRIPTION : This program implements a multilayer  *
--   *               perceptron to perform classification. *
--   *                                                     *
--   * INPUTS      : NUMBER OF NODES IN EACH LAYER         *
--   *               NUMBER OF INPUTS                      *
--   *               NUMBER OF CLASSES                     *
--   *               GAIN, GAIN DECAY AND MOMENTUM VALUES  *
--   *               COST THRESHOLD                        *
--   *               TRAINING AND TEST FILES               *
--   *                                                     *
--   * OUTPUT      : CLASSIFICATION VALUES FOR BOTH THE    *
--   *               TRAINING AND TEST FILES               *
--   *               ENDING WEIGHTS AND THRESHOLDS         *
--   ******************************************************

with text_io;                       use text_io;
with integer_text_io;               use integer_text_io;
with float_text_io;                 use float_text_io;
with vector_operations;             use vector_operations;
with perceptron_support;            use perceptron_support;
with math_lib_extension;            use math_lib_extension;
with float_math_lib;                use float_math_lib;
with system;

procedure MP is

type layer ( inputs : positive; outputs : positive ) is
    record
      X          : vector (1..inputs);              -- input to layer
      W          : matrix (1..inputs, 1..outputs); -- current weights
      W_old      : matrix (1..inputs, 1..outputs); -- previous weights
      Xp         : vector (1..outputs);             -- input to a node
      theta      : vector (1..outputs);             -- current node threshold
      theta_old  : vector (1..outputs);             -- previous node threshold
      Y          : vector (1..outputs);             -- output of layer
      del        : vector (1..outputs);             -- relative change in
    end record;                                     -- weights and thresholds

N               : integer;              -- number of features
K               : integer;              -- nodes in Layer 1
L               : integer;              -- nodes in Layer 2
M               : integer;              -- number of classes
num             : integer;              -- number of training files
file_list       : string(1..80);        -- name of file containing list
                                        -- of files


cost            : FLOAT;
total_cost      : FLOAT;
cost_threshold  : FLOAT;
output_interval: natural;
count           : natural;




center          : constant := 0.0;
width           : float;
seed            : system.unsigned_longword := 1;
Int_file        : Text_io.file_type;
training_count  : positive;
eta_decay       : FLOAT;
initial_eta     : FLOAT;
```

```
eta             : FLOAT;
alpha           : FLOAT;
correct         : float;
last            : natural;



begin  -- MAIN

  -- [0] : Get parameters from operator

  -- cost_threshold  : relative change in weights and theta values
  --                   determines when to stop training
  -- initial_eta     : gain value
  -- alpha           : momentum value
  -- file_list       : file name that contains complete list of
  --                   files to work with.  contains first, all the
  --                   training files and then all the test files
  -- output_interval : specifies the number of training iterations
  --                   between each system test
  -- width           : desired distribution radius of initial weights

  new_line;
  put("Enter cost threshold  (float) "); get (cost_threshold); skip_line;
  new_line;
  put("Enter INITIAL ETA  (float) "); get (initial_eta); skip_line;
  new_line;
  put("Enter ETA DECAY (float) "); get (eta_decay); skip_line;
  new_line;
  put("Enter ALPHA (float) "); get (alpha); skip_line;
  new_line;
  put("Enter width of distributions (float)  "); get(width); skip_line;
  new_line;
  put("Enter the number of Layer 1 nodes (K) (Int) "); get(K); skip_line;
  new_line;
  put("Enter the number of Layer 2 nodes (L) (Int) "); get(L); skip_line;
  new_line;
  put("Enter the number of input features (N) (Int) "); get(N); skip_line;
  new_line;
  put("Enter the number of output classes (M) (Int) "); get(M); skip_line;
  new_line;
  put("Enter the file list name ... ");get_line(file_list,last);
  new_line;
  put("Enter the number of training files  ... ");get(num);skip_line;
  new_line;
  put("Enter the output interval... ");get(output_interval);skip_line;


  Create ( File => Int_file,          -- if quit.test is deleted from the
           Mode => out_file,          -- directory during program execution
           Name => "Quit.test");      -- the current weights and thresholds
  Close (Int_file);                   -- will be written to a file without




                                      -- halting execution

  -- declare the network layer variables
  declare

    L1              : layer ( N, K );
    L2              : layer ( K, L );
    L3              : layer ( L, M );
    D_out           : vector ( 1..M );   -- desired output
    AO              : vector ( 1..M );   -- actual output
    X_center        : vector ( 1 .. N);  -- center of input training components
    X_width         : vector ( 1 .. N);  -- distribution width of each component

    procedure compute_output is
```

```
    begin

      L1.Xp := L1.X*L1.W;
      L1.Y  := sigmoid ( L1.Xp-L1.theta );
      L2.X  := L1.Y;

      L2.Xp := L2.X*L2.W;
      L2.Y  := sigmoid ( L2.Xp-L2.theta );
      L3.X  := L2.Y;

      L3.Xp := L3.X*L3.W;
      L3.Y  := sigmoid ( L3.Xp-L3.theta );

    end compute_output;

procedure test_data (x_center,x_width : in vector;
                     file_list        : in string;
                     last             : in natural;
                     N,M              : in integer ) is


correct     : integer := 0;
class       : integer ;
last2       : natural;
row1,col1   : integer;
Int_file    : Text_io.file_type;
Int_file2   : Text_io.file_type;
file_name   : string(1..80);
storage_array : array(1..21,1..21)of float;
x_count     : integer := 1;
d_size      : integer;
sum         : float := 0.0;
X           : vector(1..N);
D_out       : vector(1..M);
AO          : vector(1..M);

-- this procedure requires the input files to be in a specific format.
-- the first record contains the class
-- remaining records are in a row, column, value format
-- for a complete 21 x 21 array.
-- the values are not normalized and not spread
-- input data is therefore read in, normalized and spread
-- before any additional computations are accomplished

begin




  open ( file => Int_file,
         mode => in_file,
         name => file_list(1..last));

while not end_of_file(int_file) loop
   D_out := (others => 0.0);
   x_count := 1;
   get_line(int_file,file_name,last2);
   open ( file => Int_file2,
          mode => in_file,
          name => file_name(1..last2));
    get(Int_file2,class);


  For row in 1 ..21 loop
     for col in 1 .. 21 loop
        get(Int_file2,row1);
        get(Int_file2,col1);
```

```
            get(Int_file2,Integer(Storage_array(row,col)));
         end loop;
      end loop;

      close (Int_file2);

   -- normalize by deviding each point by the square root of the
   -- sum of the squares of the desired number of input points

      d_size := integer(sqrt(float(N)))/2;
      For row in 11 - d_size .. 11 + d_size loop
         For col in 11 - d_size .. 11 + d_size loop
            sum := sum + storage_array(row,col)**2;
         end loop;
      end loop;

      sum := sqrt(sum);

   -- spread the normalized input data

      For row in 11 - d_size .. 11 + d_size loop
         For col in 11 - d_size .. 11 + d_size loop
           L1.x(x_count) := (storage_array(row,col)/sum - x_center(x_count))/
                            x_width(x_count) + 0.5;
            x_count := x_count + 1;
         end loop;
      end loop;

      D_out (class) := 1.0;
      compute_output;
      AO := find_max(L3.Y);

      new_line;
      For j in L3.Y'range loop
         put(l3.y(j),4,4,0);
      end loop;

      if D_out * AO = 1.0 then
         correct := correct + 1; -- correct is based on the correct




      end if;                        -- node having the largest value
   end loop;

   new_line;

   put("Correct =  ");put(correct,6);
   new_line;

   close(int_file);
   end test_data;

      begin -- DECLARE block

       -- find the centers and width distributions of the
       -- individual components of the training data

      Get_norm_values(x_center,x_width,file_list,last,N,num);

       -- [1] : Initialize weights and thresholds for each layer

       for j in L1.W'range(2) loop
         for i in L1.W'range(1) loop
           uniform ( center, width, seed, L1.W(i,j) );
         end loop;
         uniform ( center, width, seed, L1.theta(j) );
       end loop;
       L1.W_old := L1.W;
       L1.theta_old := L1.theta;
```

```
for j in L2.W'range(2) loop
  for i in L2.W'range(1) loop
    uniform ( center, width, seed, L2.W(i,j) );
  end loop;
  uniform ( center, width, seed, L2.theta(j) );
end loop;
L2.W_old := L2.W;
L2.theta_old := L2.theta;

for j in L3.W'range(2) loop
  for i in L3.W'range(1) loop
    uniform ( center, width, seed, L3.W(i,j) );
  end loop;
  uniform ( center, width, seed, L3.theta(j) );
end loop;
L3.W_old := L3.W;
L3.theta_old := L3.theta;

-- training loop

total_cost := cost_threshold + 1.0;
count := 0;
while total_cost > cost_threshold loop

  eta := initial_eta * exp( -eta_decay * FLOAT(count) );

  -- [2] : Get input data

  get_input_data ( L1.X,D_out,x_center,x_width,num,N,file_list,last,seed );

  -- [3] : Compute Network Output




  compute_output;

  -- [4] : Update network weights

  L3.del := output_del ( L3.Y, D_out );
  update_weights ( L3.W, L3.W_old, L3.del, L3.X, eta, alpha, cost );
  total_cost := cost;

  L2.del := internal_del ( L2.Y, L3.W, L3.del );
  update_weights ( L2.W, L2.W_old, L2.del, L2.X, eta, alpha, cost );
  total_cost := total_cost + cost;

  L1.del := internal_del ( L1.Y, L2.W, L2.del );
  update_weights ( L1.W, L1.W_old, L1.del, L1.X, eta, alpha, cost );
  total_cost := total_cost + cost;

  -- [4a] Update the network thresholds
  --      threshold values of output layer are not updated

  update_thresholds ( L2.theta, L2.theta_old, L2.del, eta, alpha, cost );
  total_cost := total_cost + cost;
  update_thresholds ( L1.theta, L1.theta_old, L1.del, eta, alpha, cost );
  total_cost := total_cost + cost;

  begin -- exception block

  if count mod output_interval = 0 then
    new_line;
    put("Total Cost = "); put(total_cost,0,4,0); new_line;
    put(count,8);
    test_data(x_center,x_width,file_list,last,N,M);
    open (File => Int_file,      -- if quit.test has been deleted
          mode => out_file,      -- a name_error exception is raised
          name => "Quit.test");
```

```
            close (Int_file);
          end if;

          count := count + 1;
        EXCEPTION
           When Name_Error =>
               save_results(L1.theta,L2.theta,L3.theta,
                            x_center,x_width,L1.W,L2.W,L3.W);
               Create ( File => Int_file,
                        Mode => out_file,
                        Name => "Quit.test");
               Close (Int_file);
               count := count + 1;
        end; -- exception block
        end loop; -- training loop

        put("final count = "); put(count,4); new_line;
        save_results(L1.theta,L2.theta,L3.theta,
                     x_center,x_width,L1.W,L2.W,L3.W);

      end; -- declare block

    end mp;
```

```
--            ****************************************
--            * PACKAGE TITLE : PERCEPTRON_SUPPORT   *
--            *                                      *
--            * AUTHORS : LT STEVE TROXEL            *
--            *           CAPT DENNIS RUCK           *
--            ****************************************

--   ******************************************************
--   * DISCRIPTION : This package contains the support    *
--   *               necessary to run MP                  *
--   *                                                    *
--   * INPUTS      : Procedure Dependent                  *
--   *                                                    *
--   * OUTPUT      : Procedure Dependent                  *
--   ******************************************************

with vector_operations;          use vector_operations;
with math_lib_extension;         use math_lib_extension;
with float_math_lib;             use float_math_lib;
with float_text_io;              use float_text_io;
with integer_text_io;            use integer_text_io;
with text_io;                    use text_io;
with system;

package perceptron_support is

  function sigmoid ( input : vector ) return vector;

  function output_del ( Y, D_out : vector ) return vector;

  function internal_del ( Y          : vector;
                          W          : matrix;
                          next_del   : vector ) return vector;

  procedure update_weights ( W      : in out matrix;
                             W_old  : in out matrix;
                             del    : vector;
                             X      : vector;
                             eta    : FLOAT;
                             alpha  : FLOAT;
                             cost   : out FLOAT );

  function find_max ( X : vector ) return vector;

  procedure get_norm_values (x_center, x_width : in out vector;
                             file_list         : in string;
                             last              : in natural;
                             N                 : in integer;
                             Num               : in integer);

  procedure get_input_data ( X, D_out : out vector;
                             X_center,x_width : in vector;
                             num,N    : in integer;
                             file_list : in string;
                             last      : in natural;
                             seed : in out system.unsigned_longword );


  procedure save_results(theta_1,theta_2,theta_3,x_center,x_width : in vector;
                         W_1,W_2,W_3 : in matrix) ;




  procedure update_thresholds ( theta     : in out vector;
                                theta_old : in out vector;
                                del       : vector;
                                eta       : FLOAT;
                                alpha     : FLOAT;
                                cost      : out FLOAT );
```

```
end perceptron_support;


package body perceptron_support is

function sigmoid ( input : vector ) return vector is

output : vector ( input'range );

-- This function inplements the sigmoid function
-- used in the output calculation

begin

  for i in input'range loop
    begin
      output(i) := 1.0/(1.0+exp(-input(i)));
    exception
      when FLOOVEMAT => output(i) := 0.0;
    end;
  end loop;

  return output;

end sigmoid;


function output_del ( Y, D_out : vector ) return vector is

del : vector ( Y'range );

begin

  for i in Y'range loop
    del (i) := Y(i)*(1.0-Y(i))*(D_out(i)-Y(i));
  end loop;

  return del;

end output_del;


function internal_del ( Y          : vector;
                        W          : matrix;
                        next_del   : vector ) return vector is

del       : vector ( Y'range );
W_slice   : vector ( W'range(2) );

begin

  for j in del'range loop




    for k in W_slice'range loop
      W_slice (k) := W(j,k);
    end loop;
    del (j) := Y(j)*(1.0-Y(j))*(next_del*W_slice);
  end loop;

  return del;

end internal_del;

function find_max ( X : vector ) return vector is

max        : FLOAT := 0.0;
max_index  : integer := X'first;
```

```
output      : vector (X'range) := (others => 0.0);

begin

  for i in X'range loop
    if X(i) > max then
      max := X(i);
      max_index := i;
    end if;
  end loop;

  output(max_index) := 1.0;

  return output;

end find_max;

procedure get_norm_values (x_center, x_width : in out vector;
                           file_list         : in string;
                           last              : in natural;
                           N                 : in integer;
                           Num               : in integer) is

x_min        : vector(1 .. N) := (others => 1.0);
x_max        : vector(1 .. N) := (others => 0.0);
class        : integer;
last_sub     : natural;
row1,col1    : integer;
Int_file     : Text_io.file_type;
Int_file2    : Text_io.file_type;
file_name    : string(1..80);
storage_array : array(1..21,1..21)of float;
x_count      : integer := 1;
d_size       : integer;
sum          : float := 0.0;

-- This procedure finds the center and width of the distribution
-- of each component in the input data. This procedure is written
-- for the specific file format used in this thesis and would
-- need modification for different file structure.

begin -- get_norm_values


  open  ( file => Int_file,
          mode => in_file,




          name => file_list(1..last));

  for j in 1 .. Num loop
    get_line(int_file,file_name,last_sub);

    open  ( file => Int_file2,
            mode => in_file,
            name => file_name(1..last_sub));

  get(Int_file2,class);

  For row in 1 ..21 loop
     for col in 1 .. 21 loop
        get(Int_file2,row1);
        get(Int_file2,col1);
        get(Int_file2,Integer(Storage_array(row,col)));
     end loop;
  end loop;

  close (Int_file2);
```

```
   d_size := integer(sqrt(float(N)))/2;

   For row in 11 - d_size .. 11 + d_size loop
      For col in 11 - d_size .. 11 + d_size loop
         sum := sum + storage_array(row,col)**2;
      end loop;
   end loop;

   sum := sqrt(sum);

-- Normalize the data

   For row in 11 - d_size .. 11 + d_size loop
      For col in 11 - d_size .. 11 + d_size loop
         storage_array(row,col) := storage_array(row,col)/sum;
      end loop;
   end loop;

   x_count := 1;

-- Find max and min values for each component

   For row in 11 - d_size .. 11 + d_size loop
      For col in 11 - d_size .. 11 + d_size loop
         If storage_array(row,col) < x_min(x_count) then
            x_min(x_count) := storage_array(row,col);
         end if;

         If storage_array(row,col) > x_max(x_count) then
            x_max(x_count) := storage_array(row,col);
         end if;

         x_count := x_count + 1;
      end loop;
   end loop;

   end loop;

-- Compute center and width of distribution

   For j in x_max'range loop
      x_width(j) := x_max(j) - x_min(j);
      x_center(j) := x_width(j) / 2.0 + x_min(j);
   end loop;

   close (int_file);
end get_norm_values;


procedure get_input_data ( X, D_out : out vector;
                           x_center,x_width : in vector;
                           num,N    : in integer;
                           file_list : in string;
                           last     : in natural;
                           seed : in out system.unsigned_longword ) is

class       : integer;
pick        : float;
last_sub    : natural;
row1,col1   : integer;
Int_file    : Text_io.file_type;
file_name   : string(1..80);
storage_array : array(1..21,1..21)of float;
x_count     : integer := 1;
d_size      : integer;
sum           float := 0.0;
```

```
-- This procedure reads in the input data and performs the
-- preprocessing necessary to send data to the actual perceptron.
-- This procedure is written for the specific file format used
-- in this thesis and would need modification for different file structure.

begin

  D_out := (others => 0.0);
  open  ( file => Int_file,
          mode => in_file,
          name => file_list(1..last));

-- Compute which training file to read

  mth_random(pick,seed);
  pick := pick * float(num) - 0.499;

  if pick > float(num) - 0.5 then
     pick := float(num - 1);
  end if;

  for  j in 0 .. integer(pick) loop
     get_line(Int_file,file_name,last_sub);
  end loop;

  close(Int_file);

  open ( file => Int_file,
         mode => in_file,
         name => file_name(1..last_sub));
  get(Int_file,class);









  For row in 1 ..21 loop
     for col in 1 .. 21 loop
        get(Int_file,row1);
        get(Int_file,col1);
        get(Int_file,Integer(Storage_array(row,col)));
     end loop;
  end loop;

  close (Int_file);

  d_size := integer(sqrt(float(N)))/2;

-- Normalize and spread the data to a unit hypercube

  For row in 11 - d_size .. 11 + d_size loop
     For col in 11 - d_size .. 11 + d_size loop
        sum := sum + storage_array(row,col)**2;
     end loop;
  end loop;

  sum := sqrt(sum);

  For row in 11 - d_size .. 11 + d_size loop
     For col in 11 - d_size .. 11 + d_size loop
        x(x_count) := (storage_array(row,col)/sum - x_center(x_count))/
                      x_width(x_count) + 0.5;
        x_count := x_count + 1;
     end loop;
  end loop;

  D_out (class) := 1.0;

end get_input_data;

procedure save_results(theta_1,theta_2,theta_3,x_center,x_width : in vector;
```

```
                        W_1,W_2,W_3 : in matrix) is

        Int_file  : Text_io.file_type;

-- This procedure saves the weights, theta values, and
-- training component distributions of the network.
-- This procedure is called when the mp program is
-- terminated by the cost function falling below threshold
-- or whenever the operator desires via the deletion of the
-- Quit.test file.

begin -- save_results


      Create   (file => Int_file,
                mode => out_file,
                name => "X_center.dat");

        for j in x_center'range loop
           put(Int_file,x_center(j));
           new_line(int_file);
        end loop;

        Close (Int_file);




      Create   (file => Int_file,
                mode => out_file,
                name => "x_width.dat");

        for j in x_width'range loop
           put(Int_file,x_width(j));
           new_line(int_file);
        end loop;

        Close (Int_file);

      Create   (file => Int_file,
                mode => out_file,
                name => "L1_theta.dat");

        for j in theta_1'range loop
           put(Int_file,theta_1(j));
           new_line(int_file);
        end loop;

        Close (Int_file);


      Create   (file => Int_file,
                mode => out_file,
                name => "L2_theta.dat");

        for j in theta_2'range loop
           put(Int_file,theta_2(j));
           new_line(int_file);
        end loop;

        Close (Int_file);


      Create   (file => Int_file,
                mode => out_file,
                name => "L3_theta.dat");

        for j in theta_3'range loop
           put(Int_file,theta_3(j));
           new_line(int_file);
```

```
        end loop;

        Close (Int_file);


      Create   (file => Int_file,
                mode => out_file,
                name => "L1_W.dat");

        for j in W_1'range(1) loop
           for k in W_1'range(2) loop
              put(Int_file,W_1(j,k));
              new_line(Int_file);
           end loop;
        end loop;

        Close (Int_file);




      Create   (file => Int_file,
                mode => out_file,
                name => "L2_W.dat");

        for j in W_2'range(1) loop
           for k in W_2'range(2) loop
              put(Int_file,W_2(j,k));
              new_line(Int_file);
           end loop;
        end loop;

        Close (Int_file);


      Create   (file => Int_file,
                mode => out_file,
                name => "L3_W.dat");

        for j in W_3'range(1) loop
           for k in W_3'range(2) loop
              put(Int_file,W_3(j,k));
              New_line(Int_file);
           end loop;
        end loop;

        Close (Int_file);
end save_results;

procedure update_weights ( W      : in out matrix;
                           W_old  : in out matrix;
                           del    : vector;
                           X      : vector;
                           eta    : FLOAT;
                           alpha  : FLOAT;
                           cost   : out FLOAT  ) is
temp          : FLOAT;
cost_sum      : FLOAT := 0.0;

begin

  for j in W'range(2) loop
    for i in W'range(1) loop
      temp := W(i,j);
      W(i,j) := W(i,j) + eta*del(j)*X(i) + alpha*(W(i,j)-W_old   j));
      W_old i,j) := temp;
      cost_sum := cost_sum + ABS( (W(i,j)-W_old(i,j))/W(i,j) );
    end loop;
  end loop;
```

```
  cost := cost_sum;

end update_weights;


procedure update_thresholds ( theta     : in out vector;
                              theta_old : in out vector;
                              del       : vector;
                              eta       : FLOAT;
                              alpha     : FLOAT;




                              cost      : out FLOAT ) is

temp          : FLOAT;
cost_sum      : FLOAT := 0.0;
t_j           : integer;

begin

  for j in theta'range loop
    t_j := j;
    temp := theta(j);
    theta (j) := theta(j) + eta*del(j) + alpha*(theta(j)-theta_old(j));
    theta_old(j) := temp;
    cost_sum := cost_sum + ABS( (theta(j)-theta_old(j))/theta(j) );
  end loop;

  cost := cost_sum;

exception

  when NUMERIC_ERROR =>
        put_line("UPDATE_THRESHOLDS : NUMERIC_ERROR info follows:");
        put("Index J = "); put(t_j,3); put(",  THETA(J) = ");
        put(theta(t_j),0,4,0); put(",  DEL(J) = "); put(del(t_j),0,4,0);
        new_line; put("THETA_OLD(J) = "); put(theta_old(t_j),0,4,0);
        put(",  COST_SUM = "); put(cost_sum,0,4,0); new_line;
        put("ETA = "); put(eta,0,4,0); put(",  ALPHA = ");
        put(alpha,0,4,0); new_line;

end update_thresholds;


end perceptron_support;
```

```
--          ****************************************
--          * PACKAGE TITLE : MATH_LIB_EXTENSIONS  *
--          *                                      *
--          * AUTHORS : LT STEVE TROXEL            *
--          *           CAPT DENNIS RUCK           *
--          ****************************************

--   ******************************************************
--   * DISCRIPTION : This package contains support        *
--   *               necessary to run MP                  *
--   *                                                    *
--   * INPUTS      : Procedure Dependent                  *
--   *                                                    *
--   * OUTPUT      : Procedure Dependent                  *
--   ******************************************************

with system;                    use system;
package math_lib_extension is

  procedure mth_random ( val : out float; seed : in out unsigned_longword );
  pragma INTERFACE ( vaxrtl, mth_random );
  pragma IMPORT_VALUED_PROCEDURE ( mth_random, "MTH$RANDOM",
                                   mechanism => (value, reference));

  procedure uniform ( center  : in float;
                      width   : in float;
                      seed    : in out unsigned_longword;
                      val     : out float );

  procedure gaussian ( mean      : in float;
                       variance  : in float;
                       seed      : in out unsigned_longword;
                       val       : out float );

end math_lib_extension;

package math_lib_extension is

  procedure mth_random ( val : out float; seed : in out unsigned_longword );
  pragma INTERFACE ( vaxrtl, mth_random );
  pragma IMPORT_VALUED_PROCEDURE ( mth_random, "MTH$RANDOM",
                                   mechanism => (value, reference));

  procedure uniform ( center  : in float;
                      width   : in float;
                      seed    : in out unsigned_longword;
                      val     : out float );

  procedure gaussian ( mean      : in float;
                       variance  : in float;
                       seed      : in out unsigned_longword;
                       val       : out float );

end math_lib_extension;
```

```
--            ************************************
--            * PACKAGE TITLE : VECTOR_OPERATIONS  *
--            *                                    *
--            * AUTHOR : LT STEVE TROXEL           *
--            *          CAPT DENNIS RUCK          *
--            ************************************

--   ******************************************************
--   * DISCRIPTION : This package contains the vector math *
--   *               support necessary to run MP           *
--   *                                                     *
--   * INPUTS      : Procedure Dependent                   *
--   *                                                     *
--   * OUTPUT      : Procedure Dependent                   *
--   ******************************************************

package vector_operations is

type vector is array ( integer range <> ) of FLOAT;

type matrix is array ( integer range <>, integer range <> ) of FLOAT;

function "*"( left   : vector;
              right  : matrix ) return vector;

function "-" ( left, right : vector ) return vector;

function "*" ( left, right : vector ) return float;

end vector_operations;


package body vector_operations is

function "*"( left   : vector;
              right  : matrix ) return vector is

sum      : FLOAT;
product  : vector ( right'range(2) );

begin

  for j in right'range(2) loop
    sum := 0.0;
    for i in right'range(1) loop
      sum := sum + left(i)*right(i,j);
    end loop;
    product(j) := sum;
  end loop;

  return product;

end "*";


function "-" ( left, right : vector ) return vector is

diff : vector ( left'range );

begin




  for i in left'range loop
    diff(i) := left(i) - right(i);
  end loop;

  return diff;
```

```
end "-";

function "*" ( left, right : vector ) return FLOAT is

sum : FLOAT := 0.0;

begin

  for i in left'range loop
    sum := sum + left(i)*right(i);
  end loop;

  return sum;

end "*";

end vector_operations;
```

```
--                ******************************
--                * PROGRAM TITLE : DATA_DIS     *
--                *                              *
--                * AUTHORS : LT STEVE TROXEL    *
--                ******************************

--    ********************************************************
--    * DISCRIPTION : THIS PROGRAM COMPUTES THE DISTANCE     *
--    *               BETWEEN A TEST ARRAY AND THE AVERAGE   *
--    *               LOCATION OF A SET OF TRAINING ARRAYS   *
--    *                                                      *
--    * INPUTS      : NUMBER OF INPUT FEATURES               *
--    *               TRAINING AND TEST FILES                *
--    *                                                      *
--    * OUTPUT      : CORRESPONDING DISTANCES                *
--    ********************************************************

with text_io;              use text_io;
with float_text_io;        use float_text_io;
with integer_text_io;      use integer_text_io;
with float_math_lib;       use float_math_lib;


Procedure Data_dis is

        Type Array_Type is array(Integer range <>, Integer range <>)of float;

        S_Array               : Array_Type(1..21,1..21);
        T_Array               : Array_Type(1..21,1..21);
        S_File_Name           : string(1..80);
        T_File_Name           : string(1..80);
        File_Name             : string(1..80);
        File_LIST             : string(1..80);
        S_Last,T_Last,n,num,
        last,last2,class,
        row1,col1,d_size      : Integer;
        S_File                : Text_io.file_type;
        T_File                : Text_io.file_type;
        INT_File              : Text_io.file_type;
        INT_File2             : Text_io.file_type;
        S_row, S_col,skip,
        S_value, T_row,
        T_col, T_value        : integer;
        S_sum, T_sum,
        Distance,sum          : float := 0.0;
        ans                   : Character;

begin --Data_dis

-- this procedure requires the input files to be in a specific format.
-- the first record contains the class
-- remaining records are in a row, column, value format
-- for a complete 21 x 21 array.

        for row in 1 .. 21 loop
           for col in 1 .. 21 loop
           t_array(row,col):= 0.0;
           s_array(row,col):= 0.0;
           end loop;
        end loop;









        new_line;
        put("Enter the file list name ....");get_line(file_list,last);
        new_line;
        put("Enter the number of training files...");get(num);skip_line;
        new_line;
        put("Enter the number of input features ...");get(n);skip_line;
```

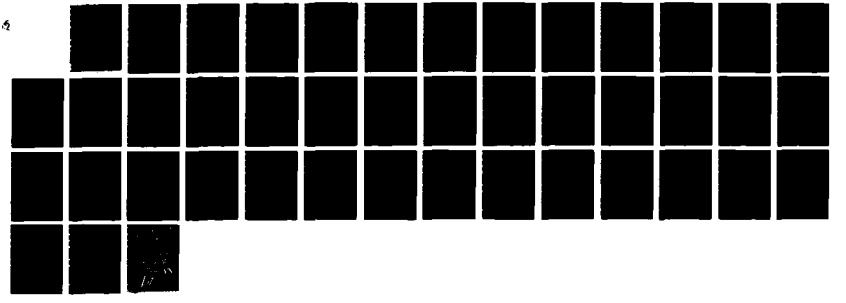
```
        d_size := integer(sqrt(float(n)))/2;
        open( file => Int_file,
              mode => in_file,
              name => file_list(1..last));

-- it is assumed that the training files are listed first in
-- the file_list file

-- find the reference point
-- this is the point specified by the average value of
-- each component in the first class of training data

for j in 1 .. num loop
           get_line(int_file,file_name,last2);

        open( file => Int_file2,
              mode => in_file,
              name => file_name(1..last2));
        get(int_file2,class);

        for row in 1 .. 21 loop
           for col in 1 .. 21 loop
              get(int_file2,row1);
              get(int_file2,col1);
              get(int_file2,integer(s_array(row,col)));
           end loop;
        end loop;
        close(int_file2);

        sum := 0.0;

-- normalize the input data

        for row in l1 - d_size .. l1 + d_size loop
           for col in l1 - d_size .. l1 + d_size loop
              sum := sum + s_array(row,col)**2;
           end loop;
        end loop;

        sum := sqrt(sum);

        for row in l1 - d_size .. l1 + d_size loop
           for col in l1 - d_size .. l1 + d_size loop
              t_array(row,col) := t_array(row,col) + s_array(row,col)/sum;
           end loop;
        end loop;

end loop;
        close(Int_file);
        for row in l1 - d_size .. l1 + d_size loop
           for col in l1 - d_size .. l1 + d_size loop
              t_array(row,col) := t_array(row,col)/float(num);









           end loop;
        end loop;


        open( file => Int_file,
              mode => in_file,
              name => file_list(1..last));

-- compute distances on the entire list of files

while not end_of_file(int_file) loop

        distance := 0.0;
        sum := 0.0;
```

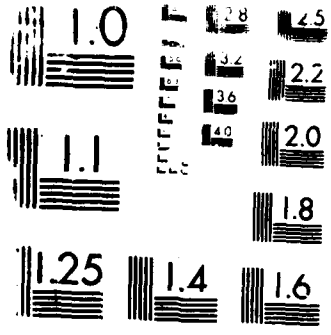
```
        file_name(1..35):="                                    ";
        get_line(int_file,file_name,last2);
        open( file => Int_file2,
              mode => in_file,
              name => file_name(1..last2));
        get(int_file2,class);

        for row in 1 .. 21 loop
           for col in 1 .. 21 loop
              get(int_file2,row1);
              get(int_file2,col1);
              get(int_file2,integer(s_array(row,col)));
           end loop;
        end loop;
        close(int_file2);


        for row in 11 - d_size .. 11 + d_size loop
           for col in 11 - d_size .. 11 + d_size loop
              sum := sum + s_array(row,col)**2;
           end loop;
        end loop;

        sum := sqrt(sum);

        for row in 11 - d_size .. 11 + d_size loop
           for col in 11 - d_size .. 11 + d_size loop
              distance := distance + (s_array(row,col)/sum -
                                      t_array(row,col))**2;
           end loop;
        end loop;
        distance := sqrt(distance);
        new_line;
        put(file_name(1..35));put(distance,4,4,0);
end loop;
        close(Int_file);
End Data_Dis;
```

AD-A188 828 POSITION SCALE AND ROTATION INVARIANT TARGET RECOGNITION USING RANGE IMAGERY(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI

UNCLASSIFIED S E TROXEL DEC 87 AFIT/GEO/ENG/87D-3 F/G 17/9 NL

2/2




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2

```
-- ************************************************************
-- * THE FOLLOWING IS A LISTING OF KOBEL AND MARTINS          *
-- * DECLARATION_HANDLER PACKAGE.  THIS PACKAGE HANDLES       *
-- * THE DECLARATIONS FOR THE EXECUTIVE PROGRAM.  EXECUTIVE   *
-- * ASSUMES A DIRECTORY SYSTEM SPECIFIED BY THE LAST_DIR     *
-- * AND DIRECTORY STATEMENTS BELOW.  THESE NEED TO BE        *
-- * CHANGED TO REFLECT THE USERS DIRECTORY SYSTEM AND        *
-- * WHERE ALL THE OUTPUTS OF EXECUTIVE ARE TO BE SENT        *
-- ************************************************************

--  Program:     DECLARATION_HANDLER
--  Authors:     Capt William Kobel and Capt Timothy Martin
--  Date:        1 October 1986
--  Language:    Vax Ada
--  System:      ISL VAX/VMS
--  Function:    Allows global declarations


package DECLARATION_HANDLER is


--  These declarations form the basic image array

      type Cmplx_Type is
          record
             Real : float;
             Imag : float;
          end record;

      type Image_Array_Type is array (integer range <>,
                                      integer range <>) of Cmplx_Type;

--  This type declaration allows three different kinds of correlation for
--   both the cyclic and linear cases

      type Corr_Flag_Type is (Cir_0, Cir_1, Cir_2, Lin_0, Lin_1, Lin_2);

--  This type declaration is used to keep track of the file types within
--  OPERATIONS_HANDLER

      type Array_Flag_Type is (Square, Lg_Square, Polar, Lg_Polar);

--  These declarations form the basic image file dimensions

      V_Row : integer := 256;
      V_Col : integer := 256;

   --  This declaration sets the default directory for all input/output
   --  procedures within FILE_HANDLER

         Last_Dir  : natural := 27;
         Directory : string (1 .. Last_Dir) := "[afituser.stroxel.vidfiles]";

-- This sets up the pragma interface to the RTL to spawn DCL commands

     procedure SEND_COMMAND (Command_Line : in string);

        pragma interface (RTL, SEND_COMMAND);

        pragma import_procedure (Internal        => SEND_COMMAND,




                                 External        => "lib$spawn",
                                 Parameter_Types => (string),
                                 Mechanism       => (descriptor(S)));


end DECLARATION_HANDLER;
```

```
--            ********************************
--            * PACKAGE TITLE : PEAK         *
--            *                              *
--            * AUTHORS : LT STEVE TROXEL    *
--            ********************************

--    ********************************************************
--    * DISCRIPTION :  THIS PROGRAM LOCATES THE PEAK VALUE   *
--    *                IN AN ARRAY AND SAVES A 21X21 ARRAY   *
--    *                OF NUMBERS AROUND THE PEAK            *
--    *                                                      *
--    * INPUTS       : 512 X 128 CORRELATION ARRAYS          *
--    *                                                      *
--    * OUTPUT       : A 21 X 21 ARRAY AROURD THE PEAK       *
--    ********************************************************

-------------------------------------------------------------------
-- THIS PROGRAM IS SET UP TO RUN IN A BATCH ENVIRONMENT          --
-- DURING THE THESIS I WAS USING EXECUTIVE TO CREATE MANY        --
-- PSRI CORRELATIONS.  THESE ARE VERY LARGE ARRAYS (2K BLOCKS)   --
-- AND TAKE TIME TO CREATE (10 MIN CPU TIME).  THIS MADE IT      --
-- NECESSARY TO RUN THE CORRELATIONS CONTINUOUSLY IN A BATCH     --
-- MODE.  HOWEVER, IN A SHORT TIME THE CREATED ARRAYS WOULD      --
-- EAT UP ALL AVAILABLE MEMOREY.  SUBMITING THIS PEAK PROGRAM    --
-- ABOUT EVERY 2 HOURS HELP IN THIS PROBLEM.                     --
-------------------------------------------------------------------

with text_io;                 use text_io;
with float_text_io;           use float_text_io;
with integer_text_io;         use integer_text_io;
with File_Handler;            use File_Handler;   -- A KOBEL,MARTIN PACKAGE
with Declaration_Handler;     use Declaration_Handler;


Procedure Peak is

        ------------------------------------------------------------
        --  For some reason these large arrays have switched to an --
        --  x,y representation where 0,0 is the lower left corner  --
        ------------------------------------------------------------

         Image                 : Image_Array_Type(0..511,0..255);
         Image_File_Name       : string(1..80);
         New_File_Name         : string(1..80);
         Total_File_Name       : string(1..Last_Dir+80);
         Internal_File         : Text_io.file_type;
         Last                  : natural := 0;
         Last_new              : natural := 0;
         Last_total            : natural := 0;
         Max_row               : integer := 0;
         Max_col               : integer := 0;
         row1,col1,row2,col2   : integer := 0;
         max_val               : float   := 0.0;
         ans                   : character;
         Extension             : string (1..4) := ".dat";

begin -- Peak

         Start_loop:
         loop




-- The peak.com file will create a file list of all
-- the .lcr files.  These files are the correlation files.
-- However, the file list will start with a directory name
-- that we don't want to read

         Get_line(Image_file_name,last);
```

```
            If Image_file_name(1) = 'R' or
               Image_file_name(1) = 'r' then
               exit Start_loop;
            end if;

            end loop start_loop;

            Peak_loop:
            loop

            max_val := 0.0;
            Read_File(Image_File_Name, Last,Image);
            Change_Extension(Image_File_Name, New_File_Name,Last,
                             Last_New, Extension);
            Last_Total := Last_new + Last_dir;
            Total_File_Name(1..Last_total) := Directory(1..Last_dir)
                                            & New_File_Name(1..Last_New);
            Create ( file => Internal_File,
                     mode => out_File,
                     name => Total_file_Name(1..(Last_Total)));

      -- find the peak value

            For row in Image'range(1) loop
               For col in Image'range(2) loop
                  If Image(row,col).real > max_val then
                     max_val := Image(row,col).real;
                     row1 := row;
                     col1 := col;
                  end if;
               end loop;
            end loop;

      -- create a 21 x 21 array around the peak
      -- a wrap around is used between the top and bottom
      -- and between the sides

            Put(row1);Put(col1);Put(max_val);
            For row in row1-10 .. row1+10 loop
               For col in col1-10 .. col1+10 loop
                  If row < 0 then
                     row2 := 512 + row;
                  else
                     if row > 511 then
                        row2 := row - 512;
                     else
                        row2 := row;
                     end if;
                  end if;
                  If col < 0 then
                     col2 := 256 + col;
                  else
                     If col > 255 then









                        col2 := col - 256;
                     else
                        col2 := col;
                     end if;
                  end if;
                  put(Internal_File,row2,6);
                  put(Internal_File,col2,6);
                  put(Internal_file,Integer(Image(row2,col2).real),10);
                  new_line(Internal_File);
               End loop;
            End loop;

            put(Internal_File,999,6);
            put(Internal_File,999,6);
```

```
        put(Internal_File,999,6);
        Close(Internal_File);

-- after the list of files there will be
-- a blank line, this is where we want to stop

        Get_line(Image_file_name,last);
        If last = 0  then
           exit peak_loop;
        end if;

        end loop peak_loop;
End Peak;
```

```
*********************************************
* THE FOLLOWING IS A LISTING OF THE PEAK.COM *
* FILE USED TO RUN THE PEAK PROGRAM          *
*********************************************

$set default [afituser.stroxel.vidfiles]
$dir/col=1/out=peak.inp *.lcr
$assign/user peak.inp sys$input
$assign/user peak.out sys$output
$run peak.exe
$del *.lcr;*
$pu *.lgf
$pu *.vid

-- The delete removes the correlation files which are not
-- needed since we now have the peak.  The pu statements are
-- to remove extra rotated versions of the psri files (.lgf)
-- and the vidio files (.vid)
```

```
--            ****************************************
--            * PACKAGE TITLE : GOODMAN_SCHWARZ      *
--            *                                      *
--            * AUTHOR : LT STEVE TROXEL             *
--            ****************************************

--    ************************************************************
--    * DISCRIPTION : This package executes a Goodman-Schwartz   *
--    *               correlation as discussed in my thesis.     *
--    *               The correlation is performed by first      *
--    *               deviding the template into 9 sections      *
--    *               and correlating with each of the 9         *
--    *               seperately.  This package also allows      *
--    *               for location selection based on the        *
--    *               number of sectors that pass threshold.     *
--    *               This locates partially occluded objects.   *
--    *                                                          *
--    * INPUTS      : Image_array, Template_array                *
--    *               Tolerance used to determine acceptable     *
--    *               peak values                                *
--    *                                                          *
--    * OUTPUT      : Location of the correlation peak           *
--    *               Complete correlation array                 *
--    ************************************************************

--    ****************************************************
--    * THIS PROCEDURE ASSUMES THAT THE SUPPLIED TEMPLATE *
--    * IS CENTERED AT THE 128,128 LOCATION IN THE ARRAY  *
--    ****************************************************

      with text_io;                      use text_io;
      with float_text_io;                use float_text_io;
      with integer_text_io;              use integer_text_io;
      with float_math_lib;               use float_math_lib;
      with declaration_handler;          use declaration_handler;
      with file_handler;                 use file_handler;--A Kobel&Martin
                                                          --  Package

Procedure Goodman_schwarz is

      Image             : Image_Array_Type(0..255,0..255);
      Temp              : Image_Array_Type(0..255,0..255);
      Temp_hold         : Image_Array_Type(0..255,0..255);
      Btmp              : Image_Array_Type(0..255,0..255);
      Num_Array         : Image_Array_Type(0..255,0..255);
      Num_sector_Array: Image_Array_Type(0..255,0..255);
      Image_name        : String(1..80);
      Temp_name         : String(1..80);
      Last_Image        : natural := 0;
      Last_Temp         : natural := 0;
      Temp_sum          : float := 0.0;
      Temp_ave          : float := 0.0;
      Temp_count        : integer := 0;
      Max_Row_Image,
      Max_Row_Temp,
      Max_Col_Image,
      Max_Col_Temp,
      S_row,
      S_col,
      Tri_row,




      Tri_col,
      Row_limit1,
      Row_limit2,
      Col_limit1,
      Col_limit2,
      Count,
      Best_row,
```

```
        Best_col,
        num_sector          : integer;
        Threshold,
        Tol,
        limit,
        Max_val,
        Sub_limit,
        Sub_thresh,
        Sub_tol             : float;
        Choice,
        Row_Dim,
        Col_Dim,
        Min_row_dim,
        Max_row_dim,
        Min_col_dim,
        Max_col_dim         : integer;
        Extension           : String(1..4) := ".bfc";
        Ans                 : Character;

-- ******************************************************
-- *            PROCEDURE FIND_DIM                      *
-- * THIS PROCEDURE FINDS THE DIMENSIONS OF THE TEMPLATE  *
-- ******************************************************

Procedure Find_dim(Temp_array       : in out Image_Array_Type;
                   Min_row_dim      : in out Integer;
                   Max_row_dim      : in out Integer;
                   Min_col_dim      : in out Integer;
                   Max_col_dim      : in out Integer) is

begin -- Find_dim

        Min_row_dim := 256;
        Max_row_dim := 0;
        Min_col_dim := 256;
        Max_col_dim := 0;

        For row in Temp_array'range(1) loop
           For col in Temp_array'range(2) loop
              If Temp_array(row,col).real /= 0.0 then
                 If row > Max_row_dim then
                    Max_row_dim := row;
                 End if;
                 If row < Min_row_dim then
                    Min_row_dim := row;
                 End if;
                 If col > Max_col_dim then
                    Max_col_dim := col;
                 End if;
                 If col < Min_col_dim then
                    Min_col_dim := col;
                 End if;
              End if;




           End loop;
        End loop;
End Find_dim;

-- ************************************************
-- *                PROCEDURE SMOOTH                *
-- * THIS PROCEDURE SMOOTHS THE VALUES OF AN ARRAY  *
-- * THIS IS NEEDED IF THE ARRAYS CONTAIN NOISE     *
-- * THAT MUST BE CORRECTED FOR                     *
-- ************************************************

-- The goal in the smoothing operation is to make sure that each
-- individual pixel is numerically between each adjacent pixel.
-- If the pixel is numerically outside the value of adjoining
```

```
-- pixels, its value is set equal to the pixel value its closest to.

Procedure Smooth (I_array  : in out Image_Array_Type) is

        T_Image        : Image_Array_Type(0..255,0..255);

Function Max(A , B : float) return float is

        C : float;

begin -- Max

        If A > B then
           C := A;
        Else
           C := B;
        End If;
        Return C;
End Max;

Function Min(A , B : float) return float is

        C : float;

begin -- Min

        If A < B then
           C := A;
        Else
           C := B;
        End If;
        Return C;
End Min;

begin -- Smooth

        T_Image := I_array;
        For row in I_array'first+1 .. I_array'last-1 loop
           For col in I_array'first+1 .. I_array'last-1 loop

              If (I_array(row,col).real >= I_array(row,col-1).real and
                  I_array(row,col).real <= I_array(row,col+1).real) or
                 (I_array(row,col).real <= I_array(row,col-1).real and
                  I_array(row,col).real >= I_array(row,col+1).real) then

                 If (I_array(row,col).real >= I_array(row-1,col).real and

                     I_array(row,col).real <= I_array(row+1,col).real) or
                    (I_array(row,col).real <= I_array(row-1,col).real and
                     I_array(row,col).real >= I_array(row+1,col).real) then

                       null;

                 elsif  I_array(row,col).real >= I_array(row-1,col).real and
                        I_array(row,col).real >= I_array(row+1,col).real then

                        T_Image(row,col).real := Max(I_array(row-1,col).real,
                                                     I_array(row+1,col).real);

                     else T_Image(row,col).real := Min(I_array(row-1,col).real,
                                                       I_array(row+1,col).real);
                 end if;

              else
                 If I_array(row,col).real >= I_array(row,col-1).real and
                    I_array(row,col).real >= I_array(row,col+1).real then

                    T_Image(row,col).real := Max(I_array(row,col-1).real,
```

```
                                                    I_array(row,col+1).real);

                        else

                           T_Image(row,col).real := Min(I_array(row,col-1).real,
                                                        I_array(row,col+1).real);
                           end if;

                        If (T_Image(row,col).real >= I_array(row-1,col).real and
                            T_Image(row,col).real <= I_array(row+1,col).real) or
                           (T_Image(row,col).real <= I_array(row-1,col).real and
                            T_Image(row,col).real >= I_array(row+1,col).real) then

                            null;

                        elsif   T_Image(row,col).real >= I_array(row-1,col).real and
                                T_Image(row,col).real >= I_array(row+1,col).real then

                            T_Image(row,col).real := Max(I_array(row-1,col).real,
                                                         I_array(row+1,col).real);

                        else

                            T_Image(row,col).real := Min(I_array(row-1,col).real,
                                                         I_array(row+1,col).real);

                        End if;
                     End if;
                  End loop;
               End loop;
               I_array := T_Image;
End Smooth;

-- ************************************************
-- *              PROCEDURE GRADIENT              *
-- * THIS PROCEDURE TAKES THE GRADIENT OF AN ARRAY *
-- ************************************************

procedure Gradient(Image_Array : in out Image_Array_Type) is




        type CX_Array_Type is array(integer range <>,integer range <>) of float;
        type CY_Array_Type is array(integer range <>,integer range <>) of float;

        sumX : float;
        sumY : float;
        CX   : CX_Array_Type(1..3,1..3);
        CY   : CY_Array_Type(1..3,1..3);
        I_hold_array : Image_array_type(0..255,0..255);

begin -- Procedure Gradient

        CX(1,1)  := -0.5;
        CX(1,2)  := -1.0;
        CX(1,3)  := -0.5;
        CX(2,1)  := 0.0;
        CX(2,2)  := 0.0;
        CX(2,3)  := 0.0;
        CX(3,1)  := 0.5;
        CX(3,2)  := 1.0;
        CX(3,3)  := 0.5;

        CY(1,1)  := -0.5;
        CY(1,2)  := 0.0;
        CY(1,3)  := 0.5;
        CY(2,1)  := -1.0;
        CY(2,2)  := 0.0;
        CY(2,3)  := 1.0;
```

```
            CY(3,1)  := -0.5;
            CY(3,2)  := 0.0;
            CY(3,3)  := 0.5;

            For row in I_hold_array'range(1) loop
               For col in I_hold_array'range(2) loop
                  I_hold_array(row,col).real := 0.0;
                  I_hold_array(row,col).imag := 0.0;
               end loop;
            end loop;


            For Row in 1..Image_Array'last(1)-1 loop
               For Col in 1..Image_Array'last(2)-1 loop
                  sumX := 0.0;
                  sumY := 0.0;

                  For n in 1..3 loop
                       For m in 1..3 loop
                          sumX := sumX +  CX(m,n) * (Image_Array(row+m-2,col+n-2).real-
                                          Image_Array(row,col).real);
                       end loop;
                  end loop;

                  For n in 1..3 loop
                       For m in 1..3 loop
                          sumY := sumY + CY(m,n) * (Image_Array(row+m-2,col+n-2).real-
                                          Image_Array(row,col).real);
                       end loop;
                  end loop;

                  If sumX > sumY then




                     I_hold_Array(row,col).real := sumX;
                  else
                     I_hold_Array(row,col).real := sumY;
                  end if;

               end loop;
            end loop;

            Image_array := I_hold_array;

      end Gradient;

      -- ******************************************************
      -- *            PROCEDURE SUB_CORR                      *
      -- * THIS PROCEDURE PERFORMS THE SUB CORRELATION        *
      -- * MULTIPLICATIONS AND ADDITIONS REQUIRED FOR EACH OF *
      -- * THE SHIFTS IN THE GOODMAN-SCHWARTZ CORRELATION     *
      -- ******************************************************

      Procedure Sub_corr (Image : in Image_Array_Type;
                          Temp  : in Image_Array_Type;
                          Btmp : in out Image_Array_Type;
                          Num_Array : in out Image_Array_Type;
                          Num_sector_Array : in out Image_Array_Type;
                          Row_limit1 : in Integer;
                          Row_limit2 : in Integer;
                          Col_limit1 : in Integer;
                          Col_limit2 : in Integer;
                          S_row      : in Integer;
                          S_col      : in Integer;
                          Row        : in Integer;
                          Col        : in Integer;
                          Count      : in out Integer;
                          Best_row   : out Integer;
                          Best_col   : out Integer;
```

```
                    Limit      : in Float;
                    Threshold  : in Float;
                    Max_val    : in out Float;
                    Choice     : in Integer;
                    Temp_ave   : in Float;
                    Temp_count : in Integer;
                    Sub_limit  : in float;
                    sub_thresh : in float) is

        Target_sum,
        Diff,
        Sum_top,
        Sum_Bottom,
        Top,
        Bottom       : Float;

Begin -- Sub_Corr

        Target_sum := 0.0;
        sum_top := 0.0;
        sum_bottom := 0.0;

   If abs(Num_Array(row,col).real-Threshold) < Limit then
        If choice = 2 then
           -- Equalize the average of the target pixels to that









           -- of the template pixels
           For Vrow in Row_limit1 .. Row_limit2  loop
              For Vcol in Col_limit1 .. Col_limit2  loop
                    Target_sum := Target_sum + Image(vrow,vcol).real *
                                  Btmp(vrow-row+128,vcol-col+128).real;
              End loop;
           End loop;
           Diff := Temp_ave - Target_sum / float(Temp_count);
        Else
           Diff := 0.0;
        End if;

        For Vrow in Row_limit1 .. Row_limit2  loop
           For Vcol in Col_limit1 .. Col_limit2  loop
              top := (Image(Vrow,Vcol).real + Diff) *
                     Temp(Vrow-row+128 , Vcol-col+128).real;
              bottom := ((Image(Vrow,Vcol).real + Diff) *
                         Btmp(Vrow-row+128,Vcol-col+128).real)**2;
              Sum_top := sum_top + top;
              Sum_bottom := sum_bottom + bottom;
           End loop;
        End loop;

        If sum_bottom /= 0.0 then
          If abs(sum_top**2/sum_bottom - sub_thresh) < sub_limit then
             Num_sector_array(row,col).real := Num_sector_array(row,col).real +
                                               1.0;
          end if;
          Num_Array(row,col).real := Num_Array(row,col).real+
                                     sum_top ** 2 / sum_bottom;
        end if;

        If Num_Array(row,col).real > Max_Val then
           Max_val := Num_Array(row,col).real;
           Best_row := row;
           Best_col := col;
        End If;

        Count := count + 1;
   else
      Num_Array(row,col).real := 0.0;
   End if;
```

```
End Sub_corr;

-- ************************************
-- * MAIN PROCEDURE OF GOODMAN_SCHWARZ *
-- ************************************

Begin -- Goodman_schwarz

        For row in Image'range(1) loop
           For col in Image'range(2) loop
              Num_Array(row,col).real := 0.0;
              Num_Array(row,col).imag := 0.0,
              Num_sector_Array(row,col).real := 0.0;
              Temp_hold(row,col).real := 0.0;
              Temp_hold(row,col).imag := 0.0;
           End loop;
        End loop;









        put("Image File: ");
        Get_File(Image_Name,Last_Image,Max_Row_Image,Max_Col_Image);
        put("Template File: ");
        Get_File(Temp_Name,Last_Temp,Max_Row_Temp,Max_Col_Temp);

--      Tol is used as what amount below threshold should a point be
--      concidered acceptable. This effects the further processing rules.
--      Sub_tol is used for each individual sector. To use this program
--      to check for partially occluded targets, tol should be set to 1.0
--      and set sub_tol to the tolerance value (ie. 0.8 means that 20% of
--      the ideal max is acceptable).

        Put("Enter Sum Tolerance : "); Get(Tol);skip_line; New_Line;
        Put("Enter Sector Tolerance : "); Get(sub_Tol);skip_line; New_Line(2);
        Put("1) Gradient Operation  "); new_line;
        Put("2) Average Equalization ");new_line(2);
        Put("Enter choice ....");get(choice);skip_line;new_line;

        Read_File(Image_Name,Last_Image,Image);
        Read_File(Temp_Name,Last_Temp,Temp);

        Smooth(Temp);
        Smooth(Image);

        If choice = 1 then

           Gradient(temp);
           Gradient(image);

           -- Elininate a two pixel boundary around the template

           For row in temp'first(1) + 2 .. temp'last(1) - 2  loop
              For col in temp'first(2) + 2 .. temp'last(2) - 2  loop
                 if temp(row,col-2).real = 0.0 or
                    temp(row,col+2).real = 0.0 or
                    temp(row-2,col).real = 0.0 or
                    temp(row+2,col).real = 0.0 then

                    temp_hold(row,col).real := 0.0;

                 else
                    temp_hold(row,col).real := temp(row,col).real;

                 end if;
              end loop;
           end loop;
```

```
   temp := temp_hold;
End if;

-- Create a binary form of the template

For row in temp'range(1) loop
   For col in temp'range(2) loop
      if temp(row,col).real = 0.0 then
         btmp(row,col).real := 0.0;
      else
         btmp(row,col).real := 1.0;









      end if;
   end loop;
end loop;

Find_dim(Temp,Min_row_dim,Max_row_dim,Min_col_dim,Max_col_dim);
row_dim := Max_row_dim - Min_row_dim + 1;
col_dim := Max_col_dim - Min_col_dim + 1;


put("Begin Correlation  ");
new_line;

S_row := Integer(row_dim /2);
S_col := Integer(col_dim /2);
Tri_row := Integer(row_dim/3);
Tri_col := Integer(col_dim/3);



For Quad in 1 .. 9 loop
   Count := 0; Max_val := 0.0; temp_sum := 0.0; temp_count := 0;
   If Quad = 1 then
      limit := 100.0; threshold := 0.0; sub_thresh := 0.0;

   -- Find the average value of the template pixels

      For row1 in 128 - S_row .. 128 - S_row + Tri_row loop
         For col1 in 128 - S_col + Tri_col + 1 ..
                     128 + S_col - Tri_col - 1 loop
            If temp(row1,col1).real /= 0.0 then
                  Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
               Temp_sum := Temp_sum + temp(row1,col1).real;
               Temp_count := Temp_count + 1;
            End if;
         End loop;
      End loop;
      Temp_ave := Temp_sum / float(temp_count);
      Sub_limit := Sub_thresh * sub_tol;

   elsif quad = 2 then

      Threshold := Threshold + sub_thresh;
      sub_thresh := 0.0;
      Put("    Threshold = ");put(threshold,2,2,0);new_line;
      limit := Threshold * Tol;

   -- Find the average value of the template pixels
      For row1 in 128 - S_row + Tri_row + 1 ..
                  128 + S_row - Tri_row - 1 loop
         For col1 in 128 - S_col + Tri_col + 1 ..
                     128 + S_col - Tri_col - 1 loop
            If temp(row1,col1).real /= 0.0 then
                  Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
               Temp_sum := Temp_sum + temp(row1,col1).real;
               Temp_count := Temp_count + 1;
            End if;
```

```
            End loop;
         End loop;
         Temp_ave := Temp_sum / float(temp_count);
         Sub_limit := Sub_thresh * sub_tol;




      elsif Quad = 3 then

         Threshold := Threshold + sub_thresh;
         sub_thresh := 0.0;
         Put("    Threshold = ");put(threshold,2,2,0);new_line;
         limit := Threshold * Tol;

      -- Find the average value of the template pixels
         For row1 in 128 + S_row - Tri_row  ..
                     128 + S_row  loop
            For col1 in 128 - S_col + Tri_col + 1 ..
                        128 + S_col - Tri_col - 1 loop
               If temp(row1,col1).real /= 0.0 then
                     Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
                  Temp_sum := Temp_sum + temp(row1,col1).real;
                  Temp_count := Temp_count + 1;
               End if;
            End loop;
         End loop;
         Temp_ave := Temp_sum / float(temp_count);
         Sub_limit := Sub_thresh * sub_tol;

      elsif Quad = 4 then

         Threshold := Threshold + sub_thresh;
         sub_thresh := 0.0;
         Put("    Threshold = ");put(threshold,2,2,0);new_line;
         limit := Threshold * Tol;


      -- Find the average value of the template pixels
         For row1 in 128 - S_row + Tri_row + 1 ..
                     128 + S_row - Tri_row - 1 loop
            For col1 in 128 - S_col  ..
                        128 - S_col + Tri_col  loop
               If temp(row1,col1).real /= 0.0 then
                     Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
                  Temp_sum := Temp_sum + temp(row1,col1).real;
                  Temp_count := Temp_count + 1;
               End if;
            End loop;
         End loop;
         Temp_ave := Temp_sum / float(temp_count);
         Sub_limit := Sub_thresh * sub_tol;

      elsif Quad = 5 then

         Threshold := Threshold + sub_thresh;
         sub_thresh := 0.0;
         Put("    Threshold = ");put(threshold,2,2,0);new_line;
         limit := Threshold * Tol;

      -- Find the average value of the template pixels
         For row1 in 128 - S_row + Tri_row + 1 ..
                     128 + S_row - Tri_row - 1 loop
            For col1 in 128 + S_col - Tri_col  ..
                        128 + S_col  loop
               If temp(row1,col1).real /= 0.0 then
                     Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
```

```
            Temp_sum := Temp_sum + temp(row1,col1).real;
            Temp_count := Temp_count + 1;
         End if;
      End loop;
   End loop;
   Temp_ave := Temp_sum / float(temp_count);
   Sub_limit := Sub_thresh * sub_tol;

elsif Quad = 6 then

   Threshold := Threshold + sub_thresh;
   sub_thresh := 0.0;
   Put("    Threshold = ");put(threshold,2,2,0);new_line;
   limit := Threshold * Tol;

-- Find the average value of the template pixels
   For row1 in 128 + S_row - Tri_row  ..
               128 + S_row  loop
      For col1 in 128 - S_col  ..
                  128 - S_col + Tri_col  loop
         If temp(row1,col1).real /= 0.0 then
               Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
            Temp_sum := Temp_sum + temp(row1,col1).real;
            Temp_count := Temp_count + 1;
         End if;
      End loop;
   End loop;
   Temp_ave := Temp_sum / float(temp_count);
   Sub_limit := Sub_thresh * sub_tol;

elsif Quad = 7 then

   Threshold := Threshold + sub_thresh;
   sub_thresh := 0.0;
   Put("    Threshold = ");put(threshold,2,2,0);new_line;
   limit := Threshold * Tol;

-- Find the average value of the template pixels
   For row1 in 128 + S_row - Tri_row  ..
               128 + S_row  loop
      For col1 in 128 + S_col - Tri_col  ..
                  128 + S_col  loop
         If temp(row1,col1).real /= 0.0 then
               Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
            Temp_sum := Temp_sum + temp(row1,col1).real;
            Temp_count := Temp_count + 1;
         End if;
      End loop;
   End loop;
   Temp_ave := Temp_sum / float(temp_count);
   Sub_limit := Sub_thresh * sub_tol;

elsif Quad = 8 then

   Threshold := Threshold + sub_thresh;
   sub_thresh := 0.0;
   Put("    Threshold = ");put(threshold,2,2,0);new_line;
   limit := Threshold * Tol;

-- Find the average value of the template pixels




   For row1 in 128 - S_row ..
               128 - S_row + Tri_row  loop
      For col1 in 128 - S_col  ..
                  128 - S_col + Tri_col loop
```

```
                If temp(row1,col1).real /= 0.0 then
                     Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
                   Temp_sum := Temp_sum + temp(row1,col1).real;
                   Temp_count := Temp_count + 1;
                End if;
             End loop;
          End loop;
          Temp_ave := Temp_sum / float(temp_count);
          Sub_limit := Sub_thresh * sub_tol;

       elsif Quad = 9 then

          Threshold := Threshold + sub_thresh;
          sub_thresh := 0.0;
          Put("    Threshold = ");put(threshold,2,2,0);new_line;
          limit := Threshold * Tol;

       -- Find the average value of the template pixels
          For row1 in 128 - S_row ..
                    128 - S_row + Tri_row  loop
             For col1 in 128 + S_col - Tri_col ..
                    128 + S_col loop
                If temp(row1,col1).real /= 0.0 then
                     Sub_thresh := sub_thresh + Temp(row1,col1).real ** 2;
                   Temp_sum := Temp_sum + temp(row1,col1).real;
                   Temp_count := Temp_count + 1;
                End if;
             End loop;
          End loop;
          Temp_ave := Temp_sum / float(temp_count);
          Sub_limit := Sub_thresh * sub_tol;

       end if;


    For Row in Image'first(1) + S_row .. Image'last(1) - S_row loop
       For Col in Image'first(2) + S_col .. Image'last(2) - S_col loop

       If Quad = 1 then
          Row_limit1 := row - S_row;
          Row_limit2 := Row_limit1 + Tri_row;
          Col_limit1 := Col - S_col + Tri_col + 1;
          Col_limit2 := Col + S_col - Tri_col - 1;

       Elsif Quad = 2 then
          Row_limit1 := Row - S_row + Tri_row + 1;
          Row_limit2 := Row + S_row - Tri_row - 1;
          Col_limit1 := Col - S_col + Tri_col + 1;
          Col_limit2 := Col + S_col - Tri_col - 1;


       Elsif Quad = 3 then
          Row_limit1 := Row + S_row - Tri_row;
          Row_limit2 := Row + S_row;
          Col_limit1 := Col - S_col + Tri_col + 1;




          Col_limit2 := Col + S_col - Tri_col - 1;

       Elsif Quad = 4 then
          Row_limit1 := Row - S_row + Tri_row + 1;
          Row_limit2 := Row + S_row - Tri_row - 1;
          Col_limit1 := Col - S_col;
          Col_limit2 := Col - S_col + Tri_col ;

       Elsif Quad = 5 then
          Row_limit1 := Row - S_row + Tri_row + 1;
          Row_limit2 := Row + S_row - Tri_row - 1;
```

```
      Col_limit1 := Col + S_col - Tri_col;
      Col_limit2 := Col + S_col ;

   Elsif Quad = 6 then
      Row_limit1 := Row + S_row - Tri_row;
      Row_limit2 := Row + S_row;
      Col_limit1 := Col - S_col;
      Col_limit2 := Col - S_col + Tri_col ;

   Elsif Quad = 7 then
      Row_limit1 := Row + S_row - Tri_row;
      Row_limit2 := Row + S_row;
      Col_limit1 := Col + S_col - Tri_col;
      Col_limit2 := Col + S_col;

   Elsif Quad = 8 then
      Row_limit1 := Row - S_row;
      Row_limit2 := Row - S_row + Tri_row;
      Col_limit1 := Col - S_col;
      Col_limit2 := Col - S_col + Tri_col;

   Elsif Quad = 9 then
      Row_limit1 := Row - S_row;
      Row_limit2 := Row - S_row + Tri_row;
      Col_limit1 := Col + S_col - Tri_col;
      Col_limit2 := Col + S_col;


   End if;

   Sub_corr (Image,Temp,Btmp,Num_Array,Num_sector_array,
             Row_limit1,Row_limit2,
             Col_limit1,Col_limit2,
             S_row,S_col,Row,Col,
             Count,Best_row,Best_col,
             limit,Threshold,Max_val,
             Choice,Temp_ave,Temp_count,
             Sub_limit,sub_thresh);

   End loop;
End loop;

put("Count = ");put(count,4);new_line;
Put(Best_row,6);Put(Best_col,6);Put(Max_val,10,2,0);

End loop;

Threshold := Threshold + Sub_thresh;
Put("    Threshold = ");put(threshold,2,2,0);new_line;




Sector_loop:
loop
put("Do you want to print the sector selections? (y/n) ");
get(ans);skip_line;new_line;
If ans = 'Y' or ans = 'y' then
   count := 0;
   put("Enter the number of sectors to threshold the print selection ");
   get(num_sector);skip_line;new_line;

   for row in num_sector_array'range(1) loop
      for col in num_sector_array'range(2) loop
         if num_sector_array(row,col).real >= float(num_sector) then
            count := count + 1;
         end if;
      end loop;
    end loop;
```

```
                put("Number of points in this selection region is ...");
                put(count,2);new_line;
                put("Do you want to print these points? (y/n)...");
                get(ans);skip_line;new_line;

                If ans = 'Y' or ans = 'y' then
                   for row in num_sector_array'range(1) loop
                     for col in num_sector_array'range(2) loop
                        if num_sector_array(row,col).real >= float(num_sector) then
                           put(row,4);put(col,4);
                           put(Integer(num_sector_array(row,col).real),4);new_line;
                        end if;
                     end loop;
                   end loop;
                end if;
             else
                exit sector_loop;
             end if;
             end loop sector_loop;

             Save_File(Image_Name,Last_Image,Num_array,Extension);

End Goodman_schwarz;
```

```
--  Program:     DISPLAY
--  Authors:     Capt William Kobel and Capt Timothy Martin
--  Date:        1 October 1986
--  Language:    Vax Ada
--  System:      ISL VAX/VMS

--  Function:    This program displays files on the Evans &
--               Southerland PS 300 raster display using a 16 bit
--               psuedocolor or greyscale.

-- This porgram was modified by Lt Troxel to include a compact data
-- procedure.  This allows for a much faster display time.  Also, the
-- display program can now be run from any terminal.

with sequential_io;
with text_io;              use text_io;
with float_text_io;        use float_text_io;
with float_math_lib;       use float_math_lib;
with integer_text_io;      use integer_text_io;
with FORTRAN_HANDLER;      use FORTRAN_HANDLER;
with FILE_HANDLER;         use FILE_HANDLER;  -- A Kobel and Martin Package
with IMAGE_HANDLER;        use IMAGE_HANDLER;  -- A Kobel and Martin Package
with DECLARATION_HANDLER; use DECLARATION_HANDLER;


------------------------------------------------------------------
--                      Procedure DISPLAY
------------------------------------------------------------------

procedure Display is

   In_File_Name : string (1 .. 80);
   Max_Row      : integer;
   Max_Col      : integer;
   Last         : natural;
   File_Error   : exception;



-----------------------------------------------
--         Procedure PROCESS_FILE
-----------------------------------------------

--  Prepares file for the Evans & Southerland raster display.
--
--  Inputs:   Max_Row  -  this must equal the row dimension of
--                        the file to be displayed.
--
--            Max_Col  -  this must equal the column dimension of
--                        the file to be displayed.

procedure PROCESS_FILE (Max_Row, Max_Col : in integer) is

   type Direction_Type is (Up, Down, Left, Right);

   type Arrow_Record_Type is
       record
          Row_Pos   : integer;
          Col_Pos   : integer;




          Direction : Direction_type;
       end record;

   type Arrow_Array_Type is array (1 .. 5) of Arrow_Record_Type;
   type Video_Flag_Type  is (Normal, Reversed, Enhanced, Original);

   Col_Boarder     : integer := 40;
```

```
   Row_Boarder      : integer := 10;
   Max_Row1         : integer := Max_Row - 1;
   Max_Col1         : integer := Max_Col - 1;
   Max_Row_Brd      : integer := Max_Row + Row_Boarder;
   Max_Col_Brd      : integer := Max_Col + Col_Boarder;
   Arrow_Array      : Arrow_Array_Type;
   Ps_Array_Color   : Ps_Array_Type (1 .. Max_Row_Brd * Max_Col_Brd, 1 .. 4);
   Image_Array      : Image_Array_Type   (0 .. Max_Row1,     0 .. Max_Col1);
   Red_Image_Array  : Display_Array_Type (1 .. Max_Row_Brd, 1 .. Max_Col_Brd);
   Grn_Image_Array  : Display_Array_Type (1 .. Max_Row_Brd, 1 .. Max_Col_Brd);
   Blu_Image_Array  : Display_Array_Type (1 .. Max_Row_Brd, 1 .. Max_Col_Brd);

   Max_Value        : float;
   File_Size        : integer;
   Threshold        : integer;
   Color_Index      : integer;
   Red              : integer;
   Green            : integer;
   Blue             : integer;
   Arrow_Count      : integer;
   Answer           : character;
   Color_Flag       : boolean;
   Arrow_Flag       : boolean;
   Video_Flag       : Video_Flag_Type;
   Intensity_Error  : exception;
   k_count          : integer;


-----------------------------------------------
--        Procedure ENHANCE_VIDEO
-----------------------------------------------

--  Enhances the lower values a file to be displayed by
--  computing the log base 10 of all values.  This is an
--  option which may be useful for files that have been
--  Fourier Transformed.
--
--  Inputs:  Image_Array  -  array which contains the un-
--                           enhanced data to be displayed.
--
--           Threshold    -  all un-enhanced values below
--                           threshold will be set to zero
--                           in the enhanced array.  This
--                           value can be used to select how
--                           low of a value to enhance.
--
--  Outputs: Image_Array  -  array which contains the enhanced
--                           data to be displayed.


procedure ENHANCE_VIDEO (Image_Array : in out Image_Array_Type;
                         Threshold   : in integer) Is




begin

   for Row in Image_Array'range(1)
   loop
      for Col in Image_Array'range(2)
      loop
         if Image_Array (Row, Col).Real < float (Threshold) then
            Image_Array (Row, Col).Real := 1.0;
         end if;
         Image_Array (Row, Col).Real := log10 ( Image_Array (Row, Col).Real);
      end loop;
   end loop;
```

```
end ENHANCE_VIDEO;



-----------------------------------------------
--          Procedure SET_COLORS
-----------------------------------------------

--  This is a color lookup table.  There are 18 colors
--  assigned; one for each of 16 possible levels,
--  one for a background color, and white which is used in
--  the scales.  The colors are formed by combining
--  different amounts of red, green, and blue pigment.
--
--  Inputs:  Color_Index  -  an integer from 0 - 15 which
--                           represents the level to be
--                           assigned a color.
--
--  Outputs: Red          -  an integer from 0 - 255 which
--                           represents the intensity of red
--                           pigment used in the color to be
--                           displayed.
--
--           Green        -  an integer from 0 - 255 which
--                           represents the intensity of green
--                           pigment used in the color to be
--                           displayed.
--
--           Blue         -  an integer from 0 - 255 which
--                           represents the intensity of blue
--                           pigment used in the color to be
--                           displayed.

procedure SET_COLORS (Color_Index      : in integer;
                      Red, Green, Blue : out integer) is

begin

   case Color_Index is
       when 0   =>  Red := 0;     Green := 0;     Blue := 110;
       when 1   =>  Red := 0 ;    Green := 0;     Blue := 150;
       when 2   =>  Red := 0;     Green := 0;     Blue := 185;
       when 3   =>  Red := 0;     Green := 0;     Blue := 220;
       when 4   =>  Red := 0;     Green := 0;     Blue := 255;




       when 5   =>  Red := 0;     Green :=  170;  Blue := 0;
       when 6   =>  Red := 0;     Green := 191;   Blue := 63;
       when 7   =>  Red := 0;     Green := 225;   Blue := 40;
       when 8   =>  Red := 0;     Green := 255;   Blue := 30;
       when 9   =>  Red := 255;   Green := 242;   Blue := 100;
       when 10  =>  Red := 255;   Green := 220;   Blue := 127;
       when 11  =>  Red := 255;   Green := 191;   Blue := 129;
       when 12  =>  Red := 255;   Green := 161;   Blue := 111;
       when 13  =>  Red := 255;   Green := 130;   Blue := 60;
       when 14  =>  Red := 230;   Green := 0;     Blue := 0;
       when 15  =>  Red := 170;   Green := 0;     Blue := 0;
       when 99  =>  Red := 0;     Green := 255;   Blue := 255;
       when 999 =>  Red := 255;   Green := 255;   Blue := 255;
       when others => null;
   end case;

end SET_COLORS;


-----------------------------------------------
```

```
--          Procedure SET_GREYSCALE
-----------------------------------------------

--  This is a Greyscale lookup table.  There are 16 grey-
--  scales assigned; one for each of 16 possible levels.
--  There is also a background color, and white which is
--  used in the scales.  The  greyscales are formed by
--  combining equal amounts of red, green, and blue pigment.
--
--  Inputs:  Color_Index  -  an integer from 0 - 15 which
--                           represents the level to be
--                           assigned a greyscale.
--
--  Outputs: Red          -  an integer from 0 - 255 which
--                           represents the intensity of red
--                           pigment used in the greyscale to
--                           be displayed.
--
--           Green        -  an integer from 0 - 255 which
--                           represents the intensity of green
--                           pigment used in the greyscale to
--                           be displayed.
--
--           Blue         -  an integer from 0 - 255 which
--                           represents the intensity of blue
--                           pigment used in the greyscale to
--                           be displayed.


procedure SET_GREYSCALE (Color_Index      : in integer;
                         Red, Green, Blue : out integer) is

begin

   case Color_Index is
       when 0   =>  Red := 90;    Green := 90;    Blue := 90;
       when 1   =>  Red := 100;   Green := 100;   Blue := 100;
       when 2   =>  Red := 110;   Green := 110;   Blue := 110;
       when 3   =>  Red := 120;   Green := 120;   Blue := 120;




       when 4   =>  Red := 130;   Green := 130;   Blue := 130;
       when 5   =>  Red := 140;   Green := 140;   Blue := 140;
       when 6   =>  Red := 150;   Green := 150;   Blue := 150;
       when 7   =>  Red := 160;   Green := 160;   Blue := 160;
       when 8   =>  Red := 170;   Green := 170;   Blue := 170;
       when 9   =>  Red := 180;   Green := 180;   Blue := 180;
       when 10  =>  Red := 190;   Green := 190;   Blue := 190;
       when 11  =>  Red := 200;   Green := 200;   Blue := 200;
       when 12  =>  Red := 210;   Green := 210;   Blue := 210;
       when 13  =>  Red := 220;   Green := 220;   Blue := 220;
       when 14  =>  Red := 230;   Green := 230;   Blue := 230;
       when 15  =>  Red := 240;   Green := 240;   Blue := 240;
       when 99  =>  Red := 0;     Green := 255;   Blue := 255;
       when 999 =>  Red := 255;   Green := 255;   Blue := 255;
       when others => null;
   end case;

end SET_GREYSCALE;


-----------------------------------------------
--          Procedure SET_BACKGROUND
-----------------------------------------------

--  Sets the background color.  The color that the
--  background is set to is determined by the color
--  settings for entry 99 in the color or greyscales
```

```
--  lookup tables.  The entire file is set to the one
--  background color; any data must be set onto the
--  background after this procedure.
--
--  Inputs:   None
--
--  Outputs:  Red_Image_Array  -  array that contains the
--                                amount of red pigment needed
--                                to create the background
--                                color.  The background color
--                                is constant so all values in
--                                this array are equal.
--
--            Grn_Image_Array  -  array that contains the
--                                amount of green pigment needed
--                                to create the background color.
--                                All values in this array are
--                                equal.
--
--            Blu_Image_Array  -  array that contains the
--                                amount of blue pigment needed
--                                to create the background color.
--                                All values in this array are
--                                equal.

procedure SET_BACKGROUND (Red_Image_Array : out Display_Array_Type;
                          Grn_Image_Array : out Display_Array_Type;
                          Blu_Image_Array : out Display_Array_Type) is

begin




   for Row in 1 .. Max_Row_Brd
   loop
      for Col in 1 .. Max_Col_Brd
      loop
         Color_Index := 99;

         if Color_Flag = false then
            SET_GREYSCALE (Color_Index, Red, Green, Blue);
         else
            SET_COLORS (Color_Index, Red, Green, Blue);
         end if;

         Red_Image_Array (Row, Col) := Red;
         Grn_Image_Array (Row, Col) := Green;
         Blu_Image_Array (Row, Col) := Blue;
      end loop;
   end loop;

end SET_BACKGROUND;



-----------------------------------------
--          Procedure SET_ARROWS       --
-----------------------------------------

--  Sets arrows on the image to be displayed.
--  The position of the arrows is determined by
--  the contents of Arrow_Array.
--
--  Inputs:   Arrow_Array      -  array that contains arrow records
--                                with information on where to place
--                                the arrows and in which direction.
--
--            Arrow_Count      -  the number of arrows to be drawn.
```

```
--
--            Red_Image_Array  -  array that contains the amount of
--                               red pigment for the image on which
--                               the arrows are to be drawn.
--
--            Grn_Image_Array  -  array that contains the amount of
--                               green pigment for the image on which
--                               the arrows are to be drawn.
--
--            Blu_Image_Array  -  array that contains the amount of
--                               blue pigment for the image on which
--                               the arrows are to be drawn.
--
--  Outputs:  Red_Image_Array  -  array that contains the amount of
--                               red pigment for the image which now
--                               has arrows placed on it.
--
--            Grn_Image_Array  -  array that contains the amount of
--                               green pigment for the image which now
--                               has arrows placed on it.
--
--            Blu_Image_Array  -  array that contains the amount of
--                               blue pigment for the image which now
--                               has arrows placed on it.



procedure SET_ARROWS (Arrow_Array     : in Arrow_Array_Type;
                      Arrow_Count     : in out integer;
                      Red_Image_Array : in out Display_Array_Type;
                      Grn_Image_Array : in out Display_Array_Type;
                      Blu_Image_Array : in out Display_Array_Type) is

   Row, Col          : integer;
   Delta_Row         : integer;
   Delta_Col         : integer;
   Col_Shift         : integer range  -1 .. 1;
   Row_Shift         : integer range  -1 .. 1;

begin

   Red   := 0;
   Green := 0;
   Blue  := 0;

   While Arrow_Count > 0
   loop


      if Arrow_Array (Arrow_Count).Direction = Up  then
         Delta_Row := 0;
         Delta_Col := -3;
         Row := Arrow_Array (Arrow_Count).Row_Pos + Row_Boarder;
         Col := Arrow_Array (Arrow_Count).Col_Pos + Col_Boarder - 7;
      elsif Arrow_Array (Arrow_Count).Direction = Down  then
         Delta_Row := 0;
         Delta_Col := 3;
         Row := Arrow_Array (Arrow_Count).Row_Pos + Row_Boarder;
         Col := Arrow_Array (Arrow_Count).Col_Pos + Col_Boarder + 7;
      elsif Arrow_Array (Arrow_Count).Direction = Left  then
         Delta_Row := 3;
         Delta_Col := 0;
         Row := Arrow_Array (Arrow_Count).Row_Pos + Row_Boarder + 7;
         Col := Arrow_Array (Arrow_Count).Col_Pos + Col_Boarder;
      elsif Arrow_Array (Arrow_Count).Direction = Right  then
         Delta_Row := -3;
         Delta_Col := 0;
         Row := Arrow_Array (Arrow_Count).Row_Pos + Row_Boarder - 7;
```

```
   Col := Arrow_Array (Arrow_Count).Col_Pos + Col_Boarder;
   Row       := Row - 3;
end if;


for Count in 1 .. 6
loop

   for Row_Shift in 0 .. 2
   loop
      for Col_Shift in 0 .. 2
      loop

         Red_Image_Array (Row + Row_Shift - 1,
                          Col + Col_Shift - 1) := Red;
         Grn_Image_Array (Row + Row_Shift - 1,
                          Col + Col_Shift - 1) := Green;
         Blu_Image_Array (Row + Row_Shift - 1,




                          Col + Col_Shift - 1) := Blue;
      end loop;
   end loop;

   if (Count = 1) and (Delta_Row = 0) then

      Red_Image_Array (Row, Col - 2 * Delta_Col / 3) := Red;
      Grn_Image_Array (Row, Col - 2 * Delta_Col / 3) := Green;
      Blu_Image_Array (Row, Col - 2 * Delta_Col / 3) := Blue;

      Red_Image_Array (Row - 2, Col) := Red;
      Grn_Image_Array (Row - 2, Col) := Green;
      Blu_Image_Array (Row - 2, Col) := Blue;

      Red_Image_Array (Row + 2, Col) := Red;
      Grn_Image_Array (Row + 2, Col) := Green;
      Blu_Image_Array (Row + 2, Col) := Blue;

      Red_Image_Array (Row + 2, Col + Delta_Col / 3) := Red;
      Grn_Image_Array (Row + 2, Col + Delta_Col / 3) := Green;
      Blu_Image_Array (Row + 2, Col + Delta_Col / 3) := Blue;

      Red_Image_Array (Row - 2, Col + Delta_Col / 3) := Red;
      Grn_Image_Array (Row - 2, Col + Delta_Col / 3) := Green;
      Blu_Image_Array (Row - 2, Col + Delta_Col / 3) := Blue;

      Red_Image_Array (Row + 3, Col + Delta_Col / 3) := Red;
      Grn_Image_Array (Row + 3, Col + Delta_Col / 3) := Green;
      Blu_Image_Array (Row + 3, Col + Delta_Col / 3) := Blue;

      Red_Image_Array (Row - 3, Col + Delta_Col / 3) := Red;
      Grn_Image_Array (Row - 3, Col + Delta_Col / 3) := Green;
      Blu_Image_Array (Row - 3, Col + Delta_Col / 3) := Blue;

   end if;

   if (Count = 1) and (Delta_Col = 0) then

      Red_Image_Array (Row - 2 * Delta_Row / 3, Col) := Red;
      Grn_Image_Array (Row - 2 * Delta_Row / 3, Col) := Green;
      Blu_Image_Array (Row - 2 * Delta_Row / 3, Col) := Blue;

      Red_Image_Array (Row, Col - 2) := Red;
      Grn_Image_Array (Row, Col - 2) := Green;
      Blu_Image_Array (Row, Col - 2) := Blue;

      Red_Image_Array (Row, Col + 2) := Red;
      Grn_Image_Array (Row, Col + 2) := Green;
      Blu_Image_Array (Row, Col + 2) := Blue;
```

```
            Red_Image_Array (Row + Delta_Row / 3, Col + 2) := Red;
            Grn_Image_Array (Row + Delta_Row / 3, Col + 2) := Green;
            Blu_Image_Array (Row + Delta_Row / 3, Col + 2) := Blue;

            Red_Image_Array (Row + Delta_Row / 3, Col - 2) := Red;
            Grn_Image_Array (Row + Delta_Row / 3, Col - 2) := Green;
            Blu_Image_Array (Row + Delta_Row / 3, Col - 2) := Blue;

            Red_Image_Array (Row + Delta_Row / 3, Col + 3) := Red;
            Grn_Image_Array (Row + Delta_Row / 3, Col + 3) := Green;




            Blu_Image_Array (Row + Delta_Row / 3, Col + 3) := Blue;

            Red_Image_Array (Row + Delta_Row / 3, Col - 3) := Red;
            Grn_Image_Array (Row + Delta_Row / 3, Col - 3) := Green;
            Blu_Image_Array (Row + Delta_Row / 3, Col - 3) := Blue;

         end if;

         Row := Row + Delta_Row;
         Col := Col + Delta_Col;

      end loop;

      Arrow_Count := Arrow_Count - 1;

   end loop;

end SET_ARROWS;


-------------------------------------------
--          Procedure SET_SCALES
-------------------------------------------

--  Creates the scales along the left and bottom borders.
--  Each grid on the scales corresponds to four pixel
--  values where a pixel corresponds to a single value
--  in the file being displayed.  The grid colors alternate
--  between two different colors with every eighth grid mark
--  displayed as a third color.  The colors that are used
--  are determined by the settings for entries 5, 8, and 999
--  in the color and greyscale lookup tables.
--
--  Inputs:   None
--
--  Outputs:  Red_Image_Array  -  array that contains the
--                                amount of red pigment needed
--                                to create the scales.
--
--            Grn_Image_Array  -  array that contains the
--                                amount of green pigment needed
--                                to create the scales.
--
--            Blu_Image_Array  -  array that contains the
--                                amount of blue pigment needed
--                                to create the scales.


procedure SET_SCALES (Red_Image_Array : in out Display_Array_Type;
                      Grn_Image_Array : in out Display_Array_Type;
                      Blu_Image_Array : in out Display_Array_Type) is

   Row_1, Row_2,
   Col_1, Col_2  : integer;
```

```
begin

   for Row_Index in 0 .. (Max_Row / 8) - 1




   loop
      for Col in (Col_Boarder - 9) .. (Col_Boarder - 4)
      loop
         for Row_Increment in (Row_Boarder + 1) .. (Row_Boarder + 4)
         loop
            Row_1 := (8 * Row_Index) + Row_Increment;
            Row_2 := Row_1 + 4;

            Color_Index := 5;

            if Color_Flag = false then
               SET_GREYSCALE (Color_Index, Red, Green, Blue);
            else
               SET_COLORS (Color_Index, Red, Green, Blue);
            end if;

            Red_Image_Array (Row_1, Col) := Red;
            Grn_Image_Array (Row_1, Col) := Green;
            Blu_Image_Array (Row_1, Col) := Blue;

            if (Row_Index + 1) rem 4  /= 0 then
               Color_Index := 8;

               if Color_Flag = false then
                  SET_GREYSCALE (Color_Index, Red, Green, Blue);
               else
                  SET_COLORS (Color_Index, Red, Green, Blue);
               end if;

               Red_Image_Array (Row_2, Col) := Red;
               Grn_Image_Array (Row_2, Col) := Green;
               Blu_Image_Array (Row_2, Col) := Blue;
            else
               Color_Index := 999;

               if Color_Flag = false then
                  SET_GREYSCALE (Color_Index, Red, Green, Blue);
               else
                  SET_COLORS (Color_Index, Red, Green, Blue);
               end if;

               Red_Image_Array (Row_2, Col) := Red;
               Grn_Image_Array (Row_2, Col) := Green;
               Blu_Image_Array (Row_2, Col) := Blue;
            end if;
         end loop;
      end loop;
   end loop;

   for Col_Index in 0 .. (Max_Col / 8) - 1
   loop
      for Row in (Row_Boarder - 9) .. (Row_Boarder - 4)
      loop
         for Col_Increment in (Col_Boarder + 1) .. (Col_Boarder + 4)
         loop
            Col_1 := (8 * Col_Index) + Col_Increment;
            Col_2 := Col_1 + 4;

            Color_Index := 5;
```

```
                  if Color_Flag = false then
                     SET_GREYSCALE (Color_Index, Red, Green, Blue);
                  else
                     SET_COLORS (Color_Index, Red, Green, Blue);
                  end if;

               Red_Image_Array (Row, Col_1) := Red;
               Grn_Image_Array (Row, Col_1) := Green;
               Blu_Image_Array (Row, Col_1) := Blue;

               if (Col_Index + 1) rem 4  /= 0 then
                  Color_Index := 8;

                  if Color_Flag = false then
                     SET_GREYSCALE (Color_Index, Red, Green, Blue);
                  else
                     SET_COLORS (Color_Index, Red, Green, Blue);
                  end if;

                  Red_Image_Array (Row, Col_2) := Red;
                  Grn_Image_Array (Row, Col_2) := Green;
                  Blu_Image_Array (Row, Col_2) := Blue;
               else
                  Color_Index := 999;

                  if Color_Flag = false then
                     SET_GREYSCALE (Color_Index, Red, Green, Blue);
                  else
                     SET_COLORS (Color_Index, Red, Green, Blue);
                  end if;

                  Red_Image_Array (Row, Col_2) := Red;
                  Grn_Image_Array (Row, Col_2) := Green;
                  Blu_Image_Array (Row, Col_2) := Blue;
               end if;
            end loop;
         end loop;
      end loop;

end SET_SCALES;


-------------------------------------------
--         Procedure SET_SPECTRUM
-------------------------------------------

--  Creates the spectrum chart for the bottom of the display.
--  The 16 colors or greyscales are displayed in ascending
--  order along the bottom border.
--
--  Inputs:  None
--
--  Outputs:  Red_Image_Array  -  array that contains the
--                                amount of red pigment needed
--                                to create the scectrum.
--
--            Grn_Image_Array  -  array that contains the
--                                amount of green pigment needed
--                                to create the spectrum.
--
--
--            Blu_Image_Array  -  array that contains the
--                                amount of blue pigment needed
--                                to create the spectrum.
```

```
procedure SET_SPECTRUM (Red_Image_Array : in out Display_Array_Type;
                        Grn_Image_Array : in out Display_Array_Type;
                        Blu_Image_Array : in out Display_Array_Type) is

   Count         : integer;
   Spectrum_Size : integer;

begin

   Spectrum_Size := Truncate (float (Max_Row) / float (16));

   Color_Index := 0;
   Count := 0;
   for Row in Row_Boarder + 1 .. Spectrum_Size * 16 + Row_Boarder
   loop
      Count := Count + 1;
      for Col in 1 .. 15
      loop

         if Color_Flag = false then
            SET_GREYSCALE (Color_Index, Red, Green, Blue);
         else
            SET_COLORS (Color_Index, Red, Green, Blue);
         end if;

         Red_Image_Array (Row, Col) := Red;
         Grn_Image_Array (Row, Col) := Green;
         Blu_Image_Array (Row, Col) := Blue;
      end loop;
      if Count = Spectrum_Size then
         Color_Index := Color_Index + 1;
         Count := 0;
      end if;
   end loop;

end SET_SPECTRUM;



------------------------------------------
--           Procedure SET_IMAGE
------------------------------------------

--  Sets the Image into the three color display arrays.
--
--  Inputs:   Image_Array  -      contains the data of the
--                                file to be displayed.
--
--  Outputs:  Red_Image_Array  -  array that contains the
--                                amount of red pigment needed
--                                to create the image.
--
--            Grn_Image_Array  -  array that contains the
--                                amount of green pigment needed




--                                to create the image.
--
--            Blu_Image_Array  -  array that contains the
--                                amount of blue pigment needed
--                                to create the image.

procedure SET_IMAGE (Image_Array     : in Image_Array_Type;
                     Red_Image_Array : in out Display_Array_Type;
                     Grn_Image_Array : in out Display_Array_Type;
                     Blu_Image_Array : in out Display_Array_Type) is
```

```
begin

   if Max_Row1 = Max_Col1 then
      for Row in reverse 0 .. Max_Row1
      loop
         for Col in 0 .. Max_Col1
         loop
            Color_Index := abs (integer (Image_Array (Max_Col1 - Col,
                                                      Max_Row1 - Row).Real));
            if Color_Flag = false then
               SET_GREYSCALE (Color_Index, Red, Green, Blue);
            else
               SET_COLORS (Color_Index, Red, Green, Blue);
            end if;

            Red_Image_Array (Max_Row_Brd - Row, Col + Col_Boarder + 1)
                                                                  := Red;
            Grn_Image_Array (Max_Row_Brd - Row, Col + Col_Boarder + 1)
                                                                  := Green;
            Blu_Image_Array (Max_Row_Brd - Row, Col + Col_Boarder + 1)
                                                                  := Blue;
         end loop;
      end loop;
   else
      for Row in 0 .. Max_Row1
      loop
         for Col in 0 .. Max_Col1
         loop
            Color_Index := abs (integer (Image_Array (Row, Col).Real));

            if Color_Flag = false then
               SET_GREYSCALE (Color_Index, Red, Green, Blue);
            else
               SET_COLORS (Color_Index, Red, Green, Blue);
            end if;

            Red_Image_Array (Row + Row_Boarder + 1, Col + Col_Boarder + 1)
                                                                  := Red;
            Grn_Image_Array (Row + Row_Boarder + 1, Col + Col_Boarder + 1)
                                                                  := Green;
            Blu_Image_Array (Row + Row_Boarder + 1, Col + Col_Boarder + 1)
                                                                  := Blue;
         end loop;
      end loop;
   end if;

end SET_IMAGE;
```

```
-----------------------------------------------
--          Compact_Data                     --
-----------------------------------------------

-- This procedure put the image into a run length encoded format
-- that allows the Evans and Sutherland to display much faster

procedure Compact_Data (Num_of_Cols   :  in out integer;
                        Num_of_Rows   :  in out integer;
                        K_Count       :  out integer;
                        Red           :  in Display_Array_Type;
                        Blue          :  in Display_Array_Type;
                        Green         :  in Display_Array_Type;
                        PS_Array_Color  : out PS_Array_Type) is

        x             : integer := red'first(1);
        y             : integer := red'first(2);
```

```
        next_x       : integer := red'first(1) + 1;
        next_y       : integer := red'first(2);
        count        : integer := 1;
        k            : integer := 1;
        num_pixels   : constant integer := ((red'last(1)-red'first(1))+1)
                                          *((red'last(2)-red'first(2))+1);

begin -- Compact_Data

        For n in 1..num_pixels-1 loop
           If (red(x,y)=red(next_x,next_y)) and
              (blue(x,y)=blue(next_x,next_y)) and
              (green(x,y)=green(next_x,next_y)) then

              count := count + 1;

           else

              PS_Array_Color(k,1) := count;
              PS_Array_Color(k,2) := red(x,y);
              PS_Array_Color(k,3) := green(x,y);
              PS_Array_Color(k,4) := blue(x,y);

              k := k+1;
              count := 1;
           end if;

           x := x + 1;
           next_x := next_x + 1;

           If x > red'last(1) then
              y := y+1;
              x := red'first(1);
              next_x := x+1;
              next_y := y;
           elsif x = red'last(1) then
              next_x := red'first(1);
              next_y := y+1;
           end if;
        end loop;

        PS_Array_Color(k,1) := count;




        PS_Array_Color(k,2) := red(x,y);
        PS_Array_Color(k,3) := green(x,y);
        PS_Array_Color(k,4) := blue(x,y);

        K_Count := k;
end Compact_Data;

-------------------------------------------
--  Main procedure for PROCESS_FILE
-------------------------------------------

begin -- PROCESS_FILE

   new_line;

   Options:
   loop
      new_line;
      put_line ("  G --> Greyscale");
      new_line;
      put_line ("  P --> Psuedocolor");
      new_line;
      put ("  Select display option => ");
      get (Answer);
      skip_line;
```

```
   case Answer is
       when 'G' | 'g' =>
           Color_Flag := False;
           exit Options;
       when 'P' | 'p' =>
           Color_Flag := True;
           exit Options;
       when others =>
           new_line (3);
           put_line ("Incorrect response!  Try again.");
           new_line;
   end case;
end loop Options;

Video:
loop
   new_line (3);
   put_line ("  N --> Normal Video");
   new_line;
   put_line ("  R --> Reverse Video");
   new_line;
   put_line ("  E --> Enhanced Video");
   new_line;
   put_line ("  O --> Original Video");
   new_line;
   put ("  Select video output => ");
   get (Answer);
   skip_line;

   case Answer is
       when 'N' | 'n' =>
           Video_Flag := Normal;
           exit Video;
       when 'R' | 'r' =>

           Video_Flag := Reversed;
           exit Video;
       when 'E' | 'e' =>
           Video_Flag := Enhanced;
           new_line (3);
           put ("  Set Threshold => ");
           get (Threshold);
           skip_line;
           new_line;
           exit Video;
       when 'O' | 'o' =>
           Video_Flag := Original;
           exit Video;
       when others    =>
           new_line (3);
           put ("You did not repond with a correct entry.");
           put_line ("  Try again.");
           new_line;

   end case;
end loop Video;

Arrows:
loop
   new_line(3);
   put_line ("  A --> Add Arrows to Dislay");
   new_line;
   put_line ("  B --> No Arrows");
   new_line;
   put ("  Select Arrow option => ");
   get (Answer);
   skip_line;
```

```
   case Answer is
       when 'A' | 'a' =>
           Arrow_Flag := true;
           new_line(3);
           put ("  Enter the number of arrows to be drawn (1 - 5) => ");
           get (Arrow_Count);

           if (Arrow_Count < 1) or (Arrow_Count > 5) then
              new_line(3);
              put ("              --  Bad Input  --");
              Wait;
              Arrow_Flag := false;
              exit Arrows;
           end if;

           for Count in 1 .. Arrow_Count
           loop
              skip_Line; new_line(3);
              put ("Arrow "); put(Count,3); new_line(2);
              put ("  Row position it should point to => ");
              get (Arrow_Array(Count).Row_Pos);
              new_line(2); skip_line;
              put ("  Col position it should point to => ");
              get (Arrow_Array(Count).Col_Pos);
              new_line(2); skip_line;
              put ("  U -> Up    D -> Down   L -> Left   R -> Right");
              new_line(2);









              put ("  Direction it should point => ");
              get (Answer);

              case Answer is
                 when 'U' | 'u' => Arrow_Array(Count).Direction := Up;
                 when 'D' | 'd' => Arrow_Array(Count).Direction := Down;
                 when 'L' | 'l' => Arrow_Array(Count).Direction := Left;
                 when 'R' | 'r' => Arrow_Array(Count).Direction := Right;
                 when others    => Arrow_Flag := false;
              end case;
           end loop;

           skip_line;
           exit Arrows;
       when 'B' | 'b' =>
           Arrow_Flag := false;
           exit Arrows;
       when others =>
           new_line (3);
           put_line ("Incorrect response!  Try again.");
           new_line;
   end case;
end loop Arrows;

new_line (4);
put ("          DISPLAYING < ");
put (In_File_Name (1 .. Last));
put_line (" >");
new_line (2);
READ_FILE (In_File_Name, Last, Image_Array);
new_line;

case Video_Flag  is
    when Normal    =>
        FIND_INTENSITY (Image_Array);
    when Reversed  =>
        FIND_REVERSE_INTENSITY (Image_Array);
    when Enhanced  =>
        ENHANCE_VIDEO (Image_Array, Threshold);
```

```
            FIND_INTENSITY (Image_Array);
         when Original =>

            -- This option allows the display of an uprocessed video
            -- file that is already represented in intensity levels
            -- between 0 and 15.

            for Row in Image_Array'range(1)
            loop
               for Col in Image_Array'range(2)
               loop
                  if Max_Value < Image_Array (Row, Col).Real then
                     Max_Value := Image_Array (Row, Col).Real;
                  end if;
               end loop;
            end loop;

            if Max_Value > 15.0 or Max_Value < 0.0 then
               raise Intensity_Error;
            end if;




        when others    =>
            new_line (3);
            put ("You did not repond with a correct entry.");
            put_line ("  Try again.");
            new_line;

    end case;

    SET_BACKGROUND (Red_Image_Array, Grn_Image_Array, Blu_Image_Array);
    SET_SCALES (Red_Image_Array, Grn_Image_Array, Blu_Image_Array);
    SET_SPECTRUM (Red_Image_Array, Grn_Image_Array, Blu_Image_Array);
    SET_IMAGE (Image_Array, Red_Image_Array, Grn_Image_Array,
                                                  Blu_Image_Array);
    if Arrow_Flag = True then
       SET_ARROWS (Arrow_Array, Arrow_Count, Red_Image_Array,
                        Grn_Image_Array, Blu_Image_Array);
    end if;

    Compact_Data (Max_Col_Brd, Max_Row_Brd, K_Count, Red_Image_Array,
                  Blu_Image_Array, Grn_Image_Array, PS_Array_Color);
    PS_RASTER_COLOR1 (Max_Col_Brd, Max_Row_Brd, K_Count, Ps_Array_Color);

    exception

       when Intensity_Error =>
          new_line (2);
          put_line ("Your data values exceed the maximum allowable intensity");
          put_line ("levels.  Your file must be normalized with respect to");
          put_line ("acceptable (0 - 15) intensity levels.");

end PROCESS_FILE;



-------------------------------------------
--  Main procedure for DISPLAY
-------------------------------------------

begin  -- DISPLAY

    new_line (24);
    put_line ("File To Be Displayed On The Evans & Southerland: ");
    new_line;
    GET_FILE (In_File_Name, Last, Max_Row, Max_Col);
    if Last = 1 and (In_File_Name (1) = 'Q' or
```

```
                    In_File_Name (1) = 'q') then
            new_line (3);
            put_line ("               ***  Leaving Display -- Good Bye !  ***");

        else

            PROCESS_FILE (Max_Row, Max_Col);

        end if;

        exception

            when File_Error =>
                new_line (12);




                put_line ("    Your file size was not [256 x 256] or [512 x 128].");
                new_line (12);

            when Name_Error =>
                new_line (12);
                put_line ("                    ****  File not in Directory  ****");
                new_line (12);


    end DISPLAY;
```

```
package FORTRAN_HANDLER is

--  This package serves as the ada-fortran interface for the
--  Evans and Sutherland PS340 raster display fortran
--  procedures used to display image arrays.


   type Ps_Array_Type is array (integer range <>,
                                integer range <>) of integer;

   type Display_Array_Type is array (integer range <>,
                                     integer range <>) of integer;


   procedure PS_RASTER_COLOR1 (Rows            : in integer;
                               Cols            : in integer;
                               K_Count         : in integer;
                               Ps_Array        : in out Ps_Array_Type);

   pragma interface (fortran, PS_RASTER_COLOR1);
   pragma import_procedure (PS_RASTER_COLOR1, mechanism => reference);

end FORTRAN_HANDLER;
```

```
c
c*******************************************************************************
c
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        subroutine PS_Raster_Color1(Num_of_Rows,Num_of_Cols,K_count,Out_Pictr)

        implicit integer*4 (X,Y)
        integer*4 Back_Gnd(3),Color_Tbl(4,256),
     +  Out_Pictr(4,k_count)

c
c [1] Position the input image on the center of the screen
c
        Xmin= (640 - Num_of_Cols)/2
        Xmax= Xmin + Num_of_Cols - 1
        Ymin= (480 - Num_of_Rows)/2
        Ymax= Ymin + Num_of_Rows - 1
c
c
c [2] Set up the RS-232 link
c
        call Pattch('logdevnam=IPS340/phydevtyp=ETHERNET',ERRHND)
c
c
c [3] Set up the WRLUT (write color look-up table)
c
c    [3.a] Set up look up table range
c
        Intns_min= 0
        Intns_max= 255
        call PRASLR(Intns_min,Intns_max,ERRHND)
c
c    [3.b] Set up the default if user does not want to set up his own table
c
        Level_Val=-1
        Level_Inc=1024/256
        Num_of_Levels=256
        do 100 J=1,Num_of_Levels,1
           Level_Val=Level_Val+Level_Inc
           Color_Tbl(1,J)=1
           Color_Tbl(2,J)=Level_Val                    ! Red Color Table
           Color_Tbl(3,J)=Level_Val                    ! Green Color Table
           Color_Tbl(4,J)=Level_Val                    ! Blue Color Table
100     continue

        Index=0
        call PRASLU(Num_of_Levels,Index,Color_Tbl,ERRHND)
c
c
c [4] set background color to light-blue
c
        Back_Gnd(1)=0
        Back_Gnd(2)=255
        Back_Gnd(3)=255
        call PRASER(Back_Gnd,ERRHND)
c
c
c [5] set the lower-left corner of the image
c




        call PRASLD(Xmin,Ymin,Xmax,Ymax,ERRHND)
c
c
c [6] display the image
c
        call PRASWP(k_count,Out_Pictr,ERRHND)
c
```

```
c
c [7] disconnect and return to host
c
        call PDTACH(ERRHND)
        return
        end
```

Bibliography

[1] Kobel, Capt William G. and Capt Timothy Martin. *Distortion Invariant Pattern Recognition in Non-Random Noise*. MS thesis, AFIT/GE/ENG/86D-20. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD- A177598).

[2] Casasent, David and others. "Real-Time Deformation Invariant Optical Pattern Recognition Using Coordinate Transformation," *Applied Optics, 26*: 938-942 (March 1987).

[3] Casasent, David and Demetri Psaltis. "Position, Rotation, and Scale Invariant Optical Correlation," *Applied Optics 15* (7): 1795-1799 (July 1976).

[4] Horev, Moshe. *Picture Correlation Model for Automatic Machine Recognition*. MS thesis, GE/EE/80D-25. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1980 (AD-A100-765).

[5] Hsu, Yuan-Neng. *et al.* "Rotation-Invariant Digital Pattern Recognition using Circular Harmonic Expansion," *Applied Optics, 21* (22): 4012-4015 (November 1982).

[6] Messner, Richard A. and Harold H. Szu. "An Image Processing Architecture for Real Time Generation of Scale and Rotation Invariant Patterns," *Computer Vision, Graghics, and Image Processing, 31* (1): 50-66 (July 1985).

[7] Nitzan, David and Richard O. Duda. "The Measurement and Use of Registered Reflectance and Range Data in Scene Analysis," *Proceedings of the IEEE 65*: 206-220 (Febuary 1977).

[8] Grantham, 2Lt Jeffrey W. *Object Recognition Using Range Images*. MS thesis, AFIT/GEP/ENP/85D-4. School of Engineering, Air Force Institute of Technology (AU), Wright- Patterson AFB OH, December 1985 (AD-A167148).

[9] Tong, 2Lt Carl W. *Target Segmentation and Image Enhancement Through Multisensor Data Fusion*. MS thesis, AFIT/GE/ENG/86D- 55. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD-A178875).

[10] Duda, Richard O. and David Nitzan. "Low-Level Processing of Registered Intensity and Range Data," *Proceedings of the Third International Conference on Pattern Recognition,*:598-601 (November 1976).

[11] Duda, Richard O., David Nitzan and Phyllis Barrett. "Use of Range and Reflectance Data to Find Planer Surface Regions," *IEEE Transactions on Pattern Recognition and Machine Intellegence, PAMI-1*:259-271 (July 1979).

[12] Rogers, Dr. Steven K., Assistant Professor. Personal interviews. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1987.

[13] Ruck, Capt Dennis W. *Multisensor Target Detection and Classification*. MS thesis, AFIT/GE/ENG/87D-56. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.

[14] Gonzalez, Rafael C. and Paul Wintz. *Digital Image Processing*. London, Amsterdam, Don Mills, Ontario, Sydney, and Tokoyo: Addison-Wesley Publishing Company, 1977.

[15] Tong, 2Lt Carl W. and others. "Multisensor Data Fusion of Laser Radar and Forward Looking Infrared (FLIR) for Target Segmentation and Enhancement," *Proceedings of the SPIE* 782:10-18 (April 1987).

[16] Mayo, 2Lt Mike W. *Computer Generated Hologram and Magneto-Optic Spatial Light Modulator for Optical Pattern Recognition*. MS thesis, AFIT/GEO/87D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.

[17] Oppenheim A.V. and J.S. Lim. "The Importance of Phase to Signal," *Proceedings of the IEEE 69*:529-541 (May 1981).

[18] Tallman, Lt Col Oliver Howard II. *The Classification of Visual Images by Spatial Filtering*. PhD Dissertation, AFIT/DS/EE/67-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1969 (AD 858866)

[19] Kabrisky, Dr. Matthew, Professor. Lecture notes taken in EENG 620, Pattern Recognition I. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1986.

[20] Goodman, J.W. *Introduction to Fourier Optics*. New York: McGraw-Hill Book Company, 1968.

[21] Vanderburg, Gordon J. and Azriel Rosenfeld. "Two-Stage Template Matching," *IEEE Transactions on Computers*, 26:384-393 (April 1977).

[22] Lippmann, Richard P. "An Introduction to Computing with Neural Networks," *IEEE ASSP Magazine, 4*:4-22 (April 1987).

[23] Ruck, Capt Dennis W., Masters Student. Personal interviews. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1987.

VITA

First Lieutenant Steven E. Troxel was born on 2 August 1959 in Minot, North Dakota. He graduated from high school in Cheyenne, Wyoming in 1977 and attended the University of Wyoming. He enlisted in the United States Air Force in December 1979 and was assigned to the 570th Strategic Missile Squadron, Davis Monthon AFB, Tucson Arizona as a Missile Combat Crew Member. He was accepted into the Airman Education and Commissioning Program in August 1981 and attended the University of Arizona from which he received the degree of Bachelor of Science in Electrical Engineering in December 1983. Upon graduation, he attended Officer Training School and was commissioned a Second Lieutenant in the United States Air Force in March 1984. After commissioning, he was stationed at F.E.Warren Air Force Base, Wyoming as a Minuteman and Peacekeeper Electrical Engineer with the Technical Engineering Branch of the 90th Strategic Missile Wing. He entered the masters program in the School of Engineering, Air Force Institute of Technology, in June 1986.

Permanent address: 501 North Main

Berthold, North Dakota

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GEO/ENG/87D-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (*If applicable*) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (*City, State, and ZIP Code*) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
POSITION, SCALE, AND ROTATION INVARIANT TARGET RECOGNITION USING RANGE IMAGERY

12. PERSONAL AUTHOR(S)
Steven E. Troxel, B.S.E.E., First Lieutenant, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1987 December	137

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Target Recognition, Pattern Recognition, Correlation, Image Processing, Learning Machines
17	11		

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Title: POSITION, SCALE,AND ROTATION INVARIANT TARGET RECOGNITION USING RANGE IMAGERY

Thesis Chairman: Dr. Steven K. Rogers, Captain, USAF

Approved for public release: IAW AFR 190-1.
31 Dec 87
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AU)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Steven K. Rogers, Captain, USAF	22b. TELEPHONE (*Include Area Code*) 513-255-6027 / 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

This thesis explores a new approach to the recognition of tactical targets using a multifunction laser radar sensor. Targets of interest were tanks, jeeps, and trucks. Doppler images were segmented and overlaided onto a relative range image. The resultant shapes were then transformed into a position, scale, and rotation invariant (PSRI) feature space. The classifiation process used the correlation peak of the template PSRI space and the target PSRI space as features. Two classification methods were implemented: a classical distance measurement approach and a new biologically-based neural network multilayer perceptron architecture.

Both methods demonstrated classification rates near 100% with a true rotation invariance demonstrated up to 20 degrees. Neural networks were shown to have a distinct advantage in a robust environment and when a figure of merit criteria was applied.

A space domain correlation was developed using local normalization and multistage processing to locate and classify targets in high clutter and with partially occluded targets.

END
DATE
FILMD
3-88
DTIC